Tuan Bui

APPLICATIONS OF MACHINE LEARNING IN EKYC'S IDENTITY DOCUMENT RECOGNITION

Information Technology

Bachelor of Engineering

2021

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree title	Time
Tuan Bui	Bachelor of Engineering	May 2021
Thesis title Applications of Machine Learning in eKYC's identity document recognition		49 pages 0 pages of appendices
Commissioned by		
Supervisor Timo Hynninen		

Abstract

Know your customer (KYC) is a client on-boarding process by which financial institutions obtain customers' information to verify their credentials. The objective of the thesis was to present the applications of Machine Learning and Deep Learning models in disrupting the traditional identity document verification task done manually in a KYC process. Financial institutions can leverage the proposed AI-based solution named eKYC's identity document recognition in this thesis to save time and incurred cost associated with the customer onboarding procedure.

The pipeline for identity document recognition solution included five modules as following: Card Detector, Card Rotator, Text Detector, Text Classifier, and Text Recognizer. Each module had its own models and each model also had its own training, evaluating, and testing process.

The proposed solution achieved an accuracy of > 98% on average, which was tested on 1000 ID images collected from public sources such as Google Images and Facebook. In terms of inference speed, the full pipeline takes a normal laptop less than one second to extract all information from an input image. Vietnamese identity documents were chosen to evaluate the solution and the results proved its applicability in real-world usages.

Keywords

Machine Learning, Deep Learning, classification, regression, object detection, optical character recognition, convolutional neural network, Random Forest, transfer learning

CONTENTS

1 INTRODUCTION

The know your customer (KYC) process, being widely used to verify the identity of new customers, has become an indispensable activity in any financial services. However, the time and incurred cost due to the outdated KYC, which requires manual verifications and face-to-face meetings, poses a burden to financial institutions, especially in developing countries. According to a survey conducted by Thompson Reuters (2016), the annual direct cost associated with the traditional KYC process is reported to be USD 60 million on average. Financial institutions, at the same time, are put at risk of postponing doing business with corporate entities and dissatisfying customers due to a lengthy and laborious onboarding procedure.

Identity document verification is undeniably among the most exasperating steps in KYC that takes most effort, time and cost. This step is, in fact, becoming more and more infuriating within the context of the coronavirus pandemic, where people are requested to practice physical distancing and face-covering in public places around the globe.

The aim of this thesis is to propose a disruptive solution to the problems involved in traditional identity document verification by using Machine Learning and Deep Learning called eKYC's identity document recognition. The disruption of eKYC's identity document recognition enables a remote, automated, quick, and simple identity verification process. The example of Vietnamese ID cards is used throughout the thesis to illustrate the application of such advanced technologies in building the modules of the solution. By choosing this example, the author would like to prove the superiority of the proposed solution in terms of its accuracy and speed.

2 BACKGROUND

This chapter discusses the prime concepts and problems which will be mentioned throughout the thesis to understand the adoption of Machine Learning and Deep Learning algorithms in eKYC's identity document recognition solution in practice.

The concepts include KYC process, identity document recognition, artificial intelligence, Machine Learning, decision tree, Random Forest, Deep Learning, transfer learning, convolutional neural network, image classification, object detection, and optical character recognition.

2.1 KYC and identity document recognition

2.1.1 The KYC process

KYC is a client-onboarding process by which financial institutions obtain customers' information to identify and verify their identity and credentials. The procedure involves identity document verification, face verification, biometric verification, and verification of other documents such as utility bills as proof of address. This mandatory process assists to avoid illegal business intentions including fraud, money-laundering activities, terrorist funding, or corruption activities. (Thales Group no date.)

Safety, speed, and accessibility are at the forefront of the financial technology revolution. However, the traditional KYC practice has its serious limitations. First, traditional account-opening and verification typically take customers 30 minutes to one hour to present themselves at a branch office and two to three days for the verification processes. Not to mention that it is required to be done during office hours on weekdays, which brings troublesomeness to the majority of the working group. Confusing and complicated paper procedures are other problems. To complete the KYC process, customers need to fill out many different papers with lots of information to be rewritten and signatures to be collected. This process requires 100% manual work, with not a single automated step. Financial institutions admit that the onboarding process is time-consuming, leaving many customers with dissatisfaction and a major weakness to overcome if they want to compete in the market. (FPT.AI, 2020.)

The severe outburst of the coronavirus pandemic, once again, proves the inefficiency in the traditional KYC process. The preventive measures of social distancing and wearing face masks in public discourages customers from having

their identity verified face-to-face. Financial institutions hence pose a pressing need for the next disruptive technology which enables the remoteness and automation of a KYC process.

2.1.2 Identity document recognition

In any KYC onboarding process, users are often requested to provide identity documents (ID) as a proof of identity and traditionally employees have to manually type out all information on these documents for later verification. Identity document recognition technology helps to automatically locate the documents and extract important information from photos captured by customers in unconstrained environments. In this thesis, the Vietnamese ID is picked out to explain the accuracy and speed of the proposed solution. Vietnam issues three types of ID cards including old soft ID, new hard ID, and the newest chip-based ID, resulting in six cases from both sides (front and back). Below are some examples of the raw image that this solution needs to extract information from.

Figure 1. Front side (left) and back side (right) of the old soft ID

Figure 2. Front side (left) and back side (right) of the new hard ID

Figure 3. Front side (left) and back side (right) of the new chip-based ‘Eid

The inconsistency of format and the variety of background, font, noise or image quality in Vietnamese ID cards present significant challenges to the identity document recognition problem that not many existing solutions can deal with.

2.2 The Artificial Intelligence revolution

Artificial intelligence (AI) has emerged with the fourth industrial revolution and has become a core component in many high-tech systems. It has crept into most areas of our modern life although we may not even realize it. Self-driving cars of Tesla, face tagging in photos by Facebook, Apple's Siri virtual assistant and Face ID, Google Translate, Google DeepMind's AlphaGo Zero, YouTube's video and Netflix’s movie recommendation systems, etc., are just a few standout examples in the vast array of artificial intelligence applications.

Machine Learning is a subset of Artificial Intelligence and a small area of Computer Science, capable of self-learning based on input data without having to be explicitly programmed. (Arthur Samuel, 1959.)

The development of computing power along with the large amounts of data collected by major technology companies like Facebook or Google has taken Machine Learning a big step further. A new field has emerged called Deep Learning. Deep Learning helped computers do what seemed impossible just ten years ago: classify and detect thousands of different objects in pictures, recognize voices, mimic human speech, translate languages, or even compose poetry or music. In other words, Deep Learning is a subset of Machine Learning and Machine Learning is a subset of Artificial Intelligence (Figure 4).

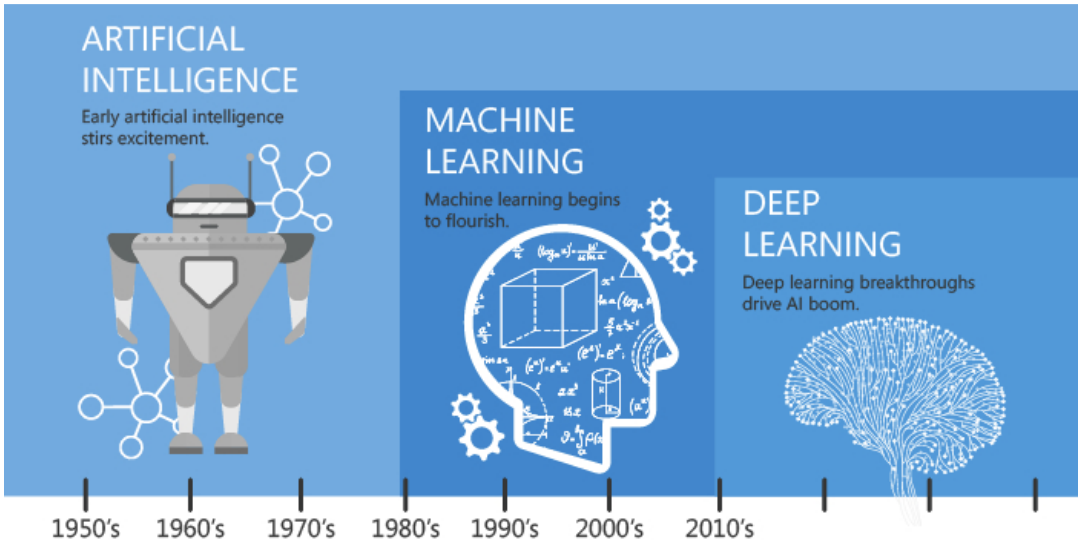

Figure 4. Relationship among Artificial Intelligence, Machine Learning, and Deep Learning (Nikhil Gupta, 2019)

2.3 Fundamentals of Machine Learning

A Machine Learning algorithm is an algorithm that learns from data. Tom Mitchell et al. (1997) gave a modern and formal definition in their book Machine Learning: "A computer program is said to learn from experience E with respect to some tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The most common way to categorize Machine Learning algorithms is based on the learning style and properties of the dataset. There are two main groups:

I. Supervised learning: The most common group of Machine Learning algorithms that predict the output y_{new} of one or more new input data x_{new} based on a learned mapping function $y_i \approx f(x_i)$, $\forall i = 1,2,...,N$. This mapping function is approximated/learned from known data pairs (x_i,y_i) or so-called (data, label) where vector $x_i \in$ a set of independent variables $X = \{x_1,x_2,...,x_N\}$ and vector $y_i \in$ a set of corresponding dependent variables $Y = \{y_1,y_2,...,y_N\}$. X x Y is also called training data. Supervised learning algorithms are further divided into two main categories:

   1. Classification: where the labels of the input data are a finite number of discrete classes (groups). For example, given a picture of a child, classify if that is a girl or a boy; given a comment of a customer from the social network, predict if that customer is happy, sad, angry, or neutral.
   2. Regression: where labels are not classified into groups but are continuous values. For example, predicting future stock prices based on historical data is a regression problem.

II. Unsupervised learning: In contrast, given large input data X as above, there is no corresponding label Y. Algorithms of this type rely on the inherent structures in the data to perform some tasks, which can also be further divided into two main categories:

   a. Clustering: A problem with a goal of grouping the given input data X into clusters based on the relationships among the data in each cluster. For example, companies segment customers into groups based on their buying behaviors to market to each group more effectively and appropriately.
   b. Association: A problem with a goal of discovering interesting and strong rules based on the given input data X. For example, using this kind of algorithm may help organizations discover that people tend to buy new furniture when they buy new houses, and then create a recommendation system exploiting these insights to push shopping demand.

The most common process that researchers and engineers follow to get a Machine Learning model that is trained on data to do some tasks, is as follow:

I. Data collecting: Gather data from diverse sources and store them. This step is very vital since the accuracy of the Machine Learning model is determined by the quantity and quality of the data.

II. Data prepararing: Manipulate raw collected data into a form that can readily and correctly be used for Machine Learning. This step involves merging and randomizing data, visualizing data, and splitting data into parts (training & evaluating). Sometimes the collected data also needs to be cleaned up by eliminating duplicates or outliers, correcting errors, filling missing values, and normalization, etc.

III. Model selecting: Select the appropriate algorithm (model) for each case and data (supervised or unsupervised learning, classification or regression problem, image data or speech data, etc.)

IV. Model training: The main and substantial step where the model is incrementally optimized (learned) by feeding a part (batch) of training data at each training iteration (step).

V. Model evaluating: This step helps to measure the accuracy of the trained model against the evaluation data (split at step 2) which is unseen by the model.

VI. Parameter Tuning: Adjusting, or tuning those so-called model hyperparameters (learning rate, number of training iterations or epochs, etc.) helps to improve the model performance further. The selection of the best hyperparameters is based on their evaluating results at step 5.

VII. Inferencing: the step where the well trained, evaluated, and chosen model is used to run inference (predict) on new data as its original purpose.

2.4 Decision Tree and Random Forest classifier

Decision Tree is a supervised learning algorithm that can be applied for both classification and regression problems, that is to say, can take two kinds of target variable: categorical variable and continuous variable. The Decision Tree model learns decision rules deduced from the training data features, which makes it able to predict the target variable’s value. When the “tree” grows deeper, decision rules

become more complex, i.e. model is fitter (Scikit-learn no date.). It is basically a series of consecutive decisions made by checking conditions based on a collection of predefined features/attributes to reach a certain Leaf/Terminal node (Figure 5).

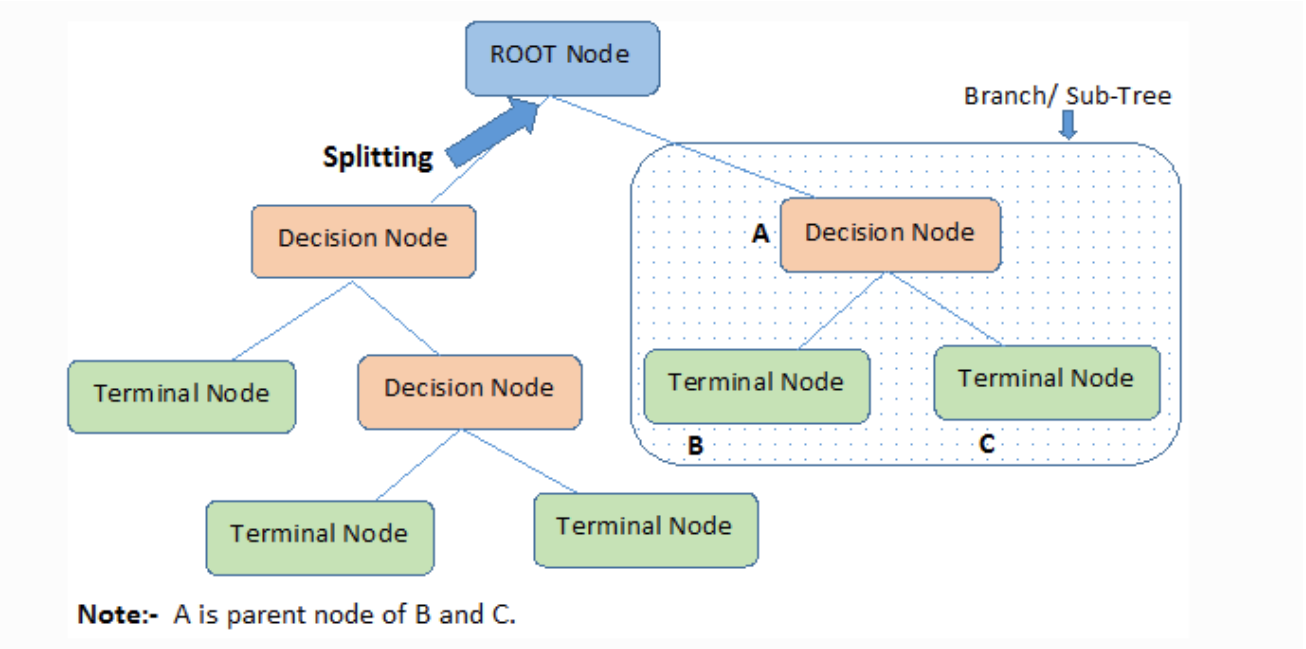

Figure 5. Topology of Decision Tree (Nagesh Singh Chauhan, 2020)

To predict the class label for the input data, the algorithm starts sorting down from the Root Node of the tree as shown in Figure 5. At this initial point, the value of the input data's feature is compared with the value of the Root Node feature, and then the algorithm recursively jumps to the sub-node (another test case for some feature) of the branch corresponding to that value. The process only ends until the algorithm reaches a Leaf/Terminal Node that does not split anymore and gives the class label for the input data. Figure 6 illustrates how a bank uses the Decision Tree with conditions from three features (credit history, income, and loan amount) to decide whether the customer's loan should be accepted.

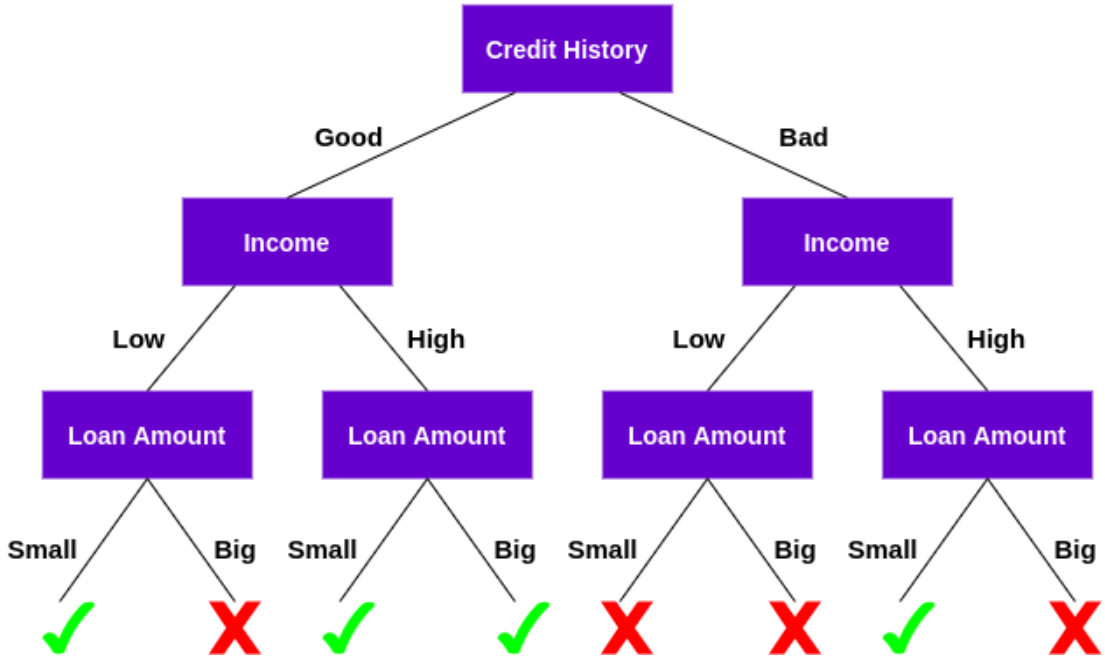

Figure 6. Example of a bank using Decision Tree for loan approval (Abhishek Sharma, 2019)

Like other algorithms and models, Decision Tree also suffers from bias and variance trade-offs. High bias is the outcome of simplifying the assumptions of the algorithm too much to make it easier to learn, then causing a model to fail to capture the key relations between features and target variables (so-called underfitting). High variance is high sensitivity to minor oscillations in the training data. It can cause an algorithm to learn and capture the outliers in the training data, instead of the intended outputs (so-called overfitting). (Wikipedia no date.)

As we can see, the Decision Tree algorithm is quite simple and intuitive. But most of the time, a single tree is not powerful enough to deal with complicated problems and often maximally overfit to the training data, i.e. the Decision Tree model performs badly on unobserved data. This is where ensemble learning algorithms like Random Forest come into play. “Ensemble” means “group”, and ensemble learning methods integrate a group of some weaker Machine Learning models into one powerful model to achieve a better accuracy as well as better balance between variance and bias (Figure 7). The base models can be any Machine Learning models such as Decision Tree, Support Vector Machines (SVM), etc. and some common types of ensembles include Bootstrap Aggregating (Bagging), Boosting and Stacking. (Wikipedia no date.)

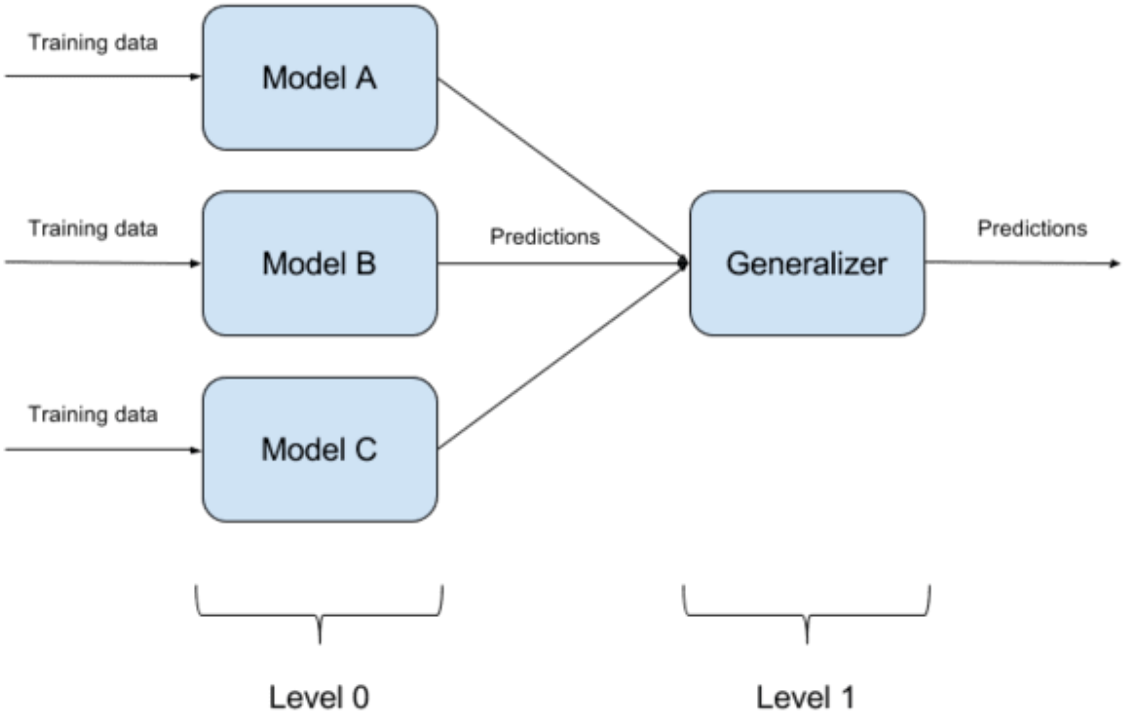

Figure 7. Combining an ensemble of Machine Learning-based models for better performance (CommonLounge, 2018)

The model that is proposed to build the Text Classifier module in section 3.1.4 - Random Forest is a supervised learning algorithm used for both classification and regression problems, and is also a type of Bagging model in ensemble learning methods. At the high level, Random Forest includes various decision trees that is learned from training data samples. In a classification problem, each individual tree provides a classification label (a vote) and Random Forest makes the final prediction by voting (select the one having the most votes over all decision trees in the forest). In a regression problem, the final prediction is the average of outputs by all trees.

Random Forest makes use of Bagging and random feature selection, and its creation process is summarized as follows:

- Firstly, assume the number of samples in the training set is N. Then, instead of the original training data, a sample of these N cases is taken at random

but with replacement to train each tree from the predefined n number of trees. This random process is called Bootstrap Aggregating or Bagging.

- Let's assume there are M features, a number m<M is selected and for every tree, m features are randomly chosen from the M original features to use. Among these m features, the one giving the best split is applied to split each node. This number m is immutable.
- Every tree is grown as much as possible like a normal decision tree but there is no need to reduce the trees' complexity by removing non-critical sections of them.
- Predict with new data (inferencing) by voting the predictions of the n trees for classification or by averaging for regression. (Analytics Vidhya, 2016.)

In the end, Random Forest ends up with an ensemble ("forest") of trees that are trained on different samples of data while also using different sets of features to make predictions. This helps to ensure that different trees protect each other from their own mistakes. Although some trees' decisions are incorrect, many others' will be accurate, so being a "forest" makes the trees able to make the aggregated decisions correctly. In other words, this ensemble learning method gives better accuracy and reduces overfitting.

2.5 Fundamentals of Deep Learning

Deep Learning is a subset of Machine Learning and its underlying architecture was motivated by the structure of a person's brain. At high level, the architecture of Deep Learning (or basically deep neural networks) is made up of many perceptrons which are computational units that take input signals and convert them into outputs signals with the support of nonlinear functions, just like the human brain's neuron transfers electrical pulses throughout the nervous system. Each perceptron itself first takes all the inputs (x_1 through x_n) and weights (w_1 through w_n, and bias term b) to compute a weighted sum Z as step 1 in Figure 8. This output is then delivered to an activation function g() which is a nonlinear function that maps the calculated sum Z into the preferred range before that mapping result is sent again as the input to another perceptron.

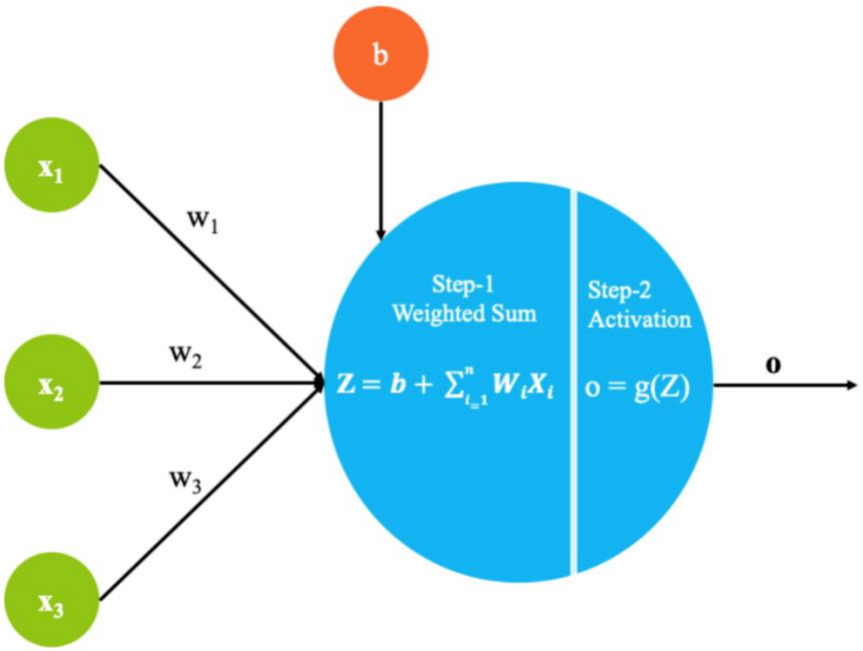

Figure 8. The weights are applied to the inputs, resulting in a weighted sum that is delivered to an activation function to produce the output (Piyush M. & Samaya M., 2020)

Many of the these perceptrons make up a shallow neural network which is a neural network having only three layers: one input layer, one hidden layer, and one output layer as shown in Figure 9.

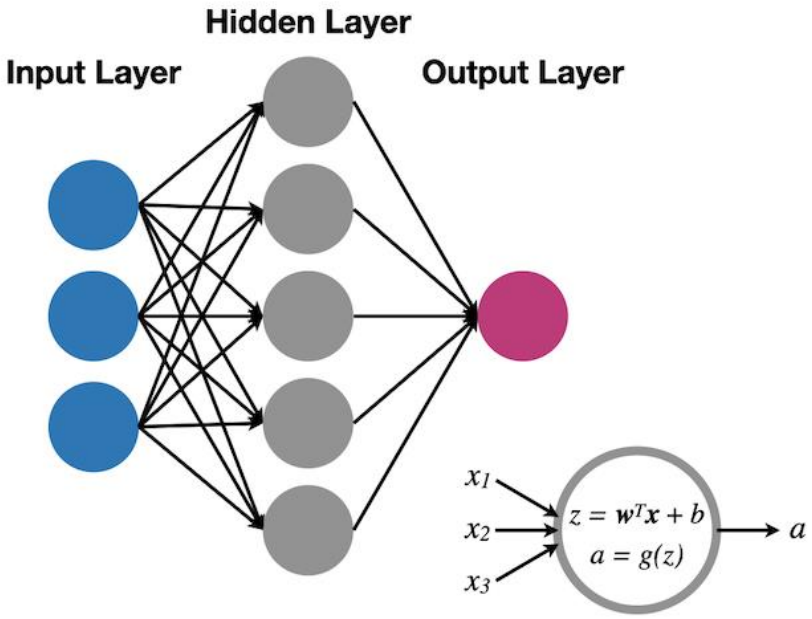

Figure 9. A shallow neural network with only one hidden layer (Christian, 2020)

Finally, a deep neural network is basically a deeper version of a shallow neural network with many hidden layers in between where neurons are connected to each other by the weights which manipulate how much each neuron's activation affects the other neurons connected to it (Figure 10). This is also a type of model that can be used to solve both regression and classification problems.

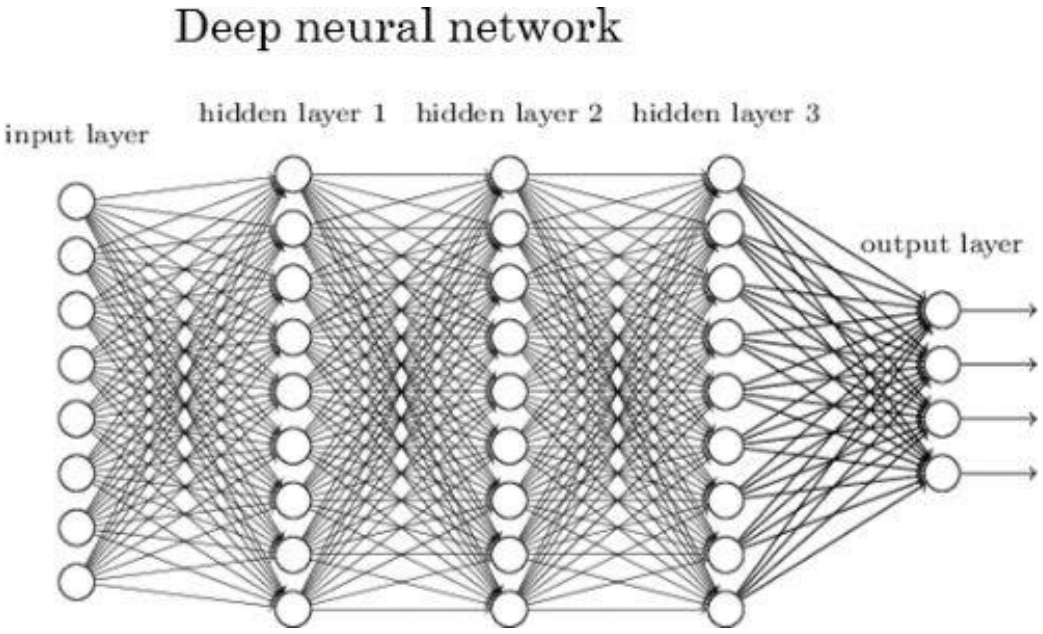

Figure 10. A deep neural network with three hidden layers (Anastasua K. & Listlink, 2020)

The above fundamental concepts of Deep Learning came from several last decades, but it has just made a breakthrough since about 6-7 years ago. There were some factors that led to this boom:

- Very large labeled datasets were generated.
- The parallel computing power was drastically enhanced with GPUs.
- The invention of ReLU and other activation functions diminished the vanishing gradient issue.
- The improvement of many state-of-the-art deep neural network architecture: RestNet, EfficientNet, etc. and the new transfer learning, fine-tuning methods.
- Numerous novel regularization and optimization methods: dropout, batch normalization, data augmentation, Adam, Adagrad, etc.

- Lots of new libraries and frameworks support Deep Learning training with GPUs: Keras, Tensorflow, Pytorch, etc.

2.6 Transfer learning and convolutional neural networks for image classification

Image classification is one of the most common tasks in Computer Vision which gives an input image one label from a list of classes. This problem has many practical applications and is also the foundation of other more advanced tasks such as object detection and image segmentation (Github no date).

Before Deep Learning, image classification (so as other classification problems) was resolved by traditional Machine Learning and often divided into two separate phases: feature engineering and training a classifier (a learning algorithm) (Figure 11). At feature engineering step, common feature descriptors used for image-related problems are Scale Invariant Feature Transform (SIFT) (David G. L., 1999), HOG (Histogram of Oriented Gradients) (Bill T. & Navneet D., 2005), and LBP (Local Binary Pattern), etc. Common Classifiers are Random Forest (as presented in section 2.4), multi-class SVM (Alex J. S., Robert C. W., Peter L., 1999), or Logistic Regression etc.

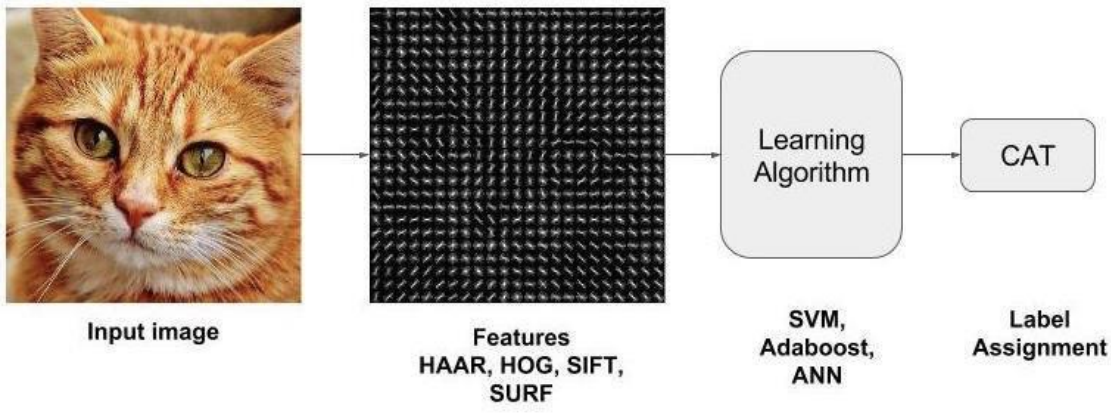

Figure 11. Image classification pipeline with traditional Machine Learning approach (Satya M., 2017)

The feature engineering methods above are hand-crafted features since they are primarily based on observations of image characteristics. They provide good

results in some cases but still have many limitations because the process of discovering the right features and classifiers is separate and very costly.

The results for the Image classification problem are progressively enhanced with the growing of Deep Learning, computing power, and huge datasets. The most common base dataset is ImageNet (https://www.image-net.org) with 1.2M images of 1000 different classes. Many Deep Learning models have outperformed the above mentioned traditional Machine Learning approaches and achieved state-of-the-art results on this dataset, e.g. VGG, ResNet, EfficientNet, etc (Paperswithcode no date).

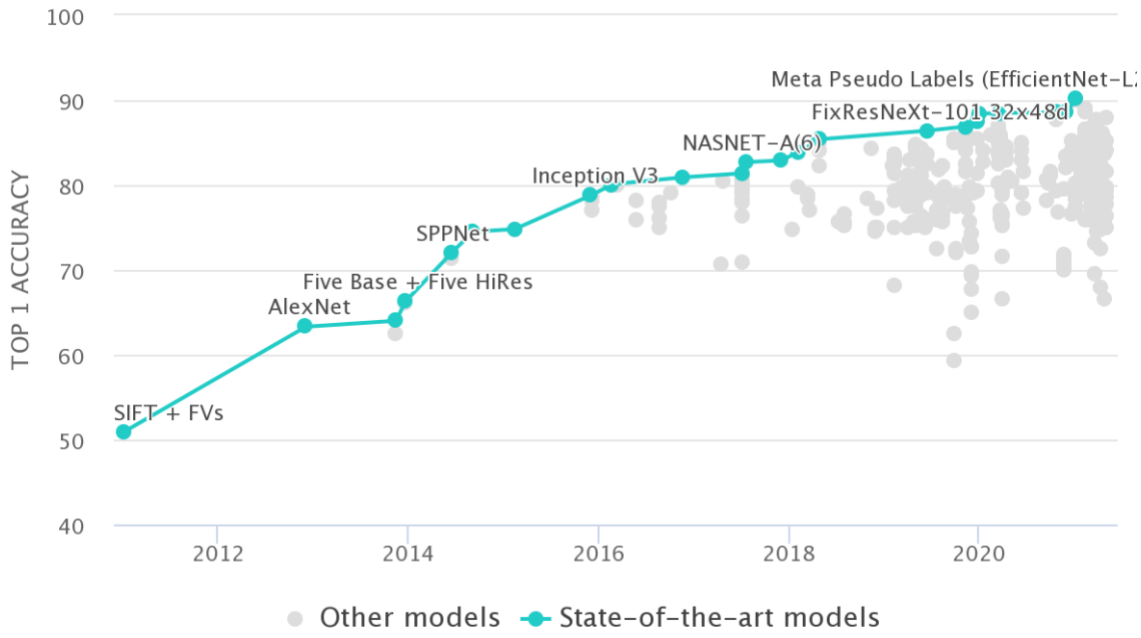

Figure 12. Image classification results on ImageNet (Paperswithcode no date)

These models are convolutional neural networks (ConvNets) that are basically ordinary deep neural networks. However, the ConvNet architecture assume that the inputs are images (3D volume with height, width, and depth), which allows to encode relevant properties to the network. ConvNet transform this 3D image volume into an output volume that provides the class scores, through many stacked layers with learnable weights. These hidden layers are usually convolutional layers and pooling layers while the last layer is a fully connected layer and is usually a Softmax Regression (Github no date) as illustrated in Figure 13. The number of

units in the last layer is equal to the number of classes (e.g. ImageNet has 1000 classes such as dog, cat, car, etc.). In the traditional Machine Learning point of view, the output at the layer before the fully connected layer can be considered as a feature vector (extractor) and Softmax Regression is a classifier. However, these features and classifiers are trained and optimized together through deep networks (in this case, ConvNet) with a very large dataset, which makes these models achieve much better results.

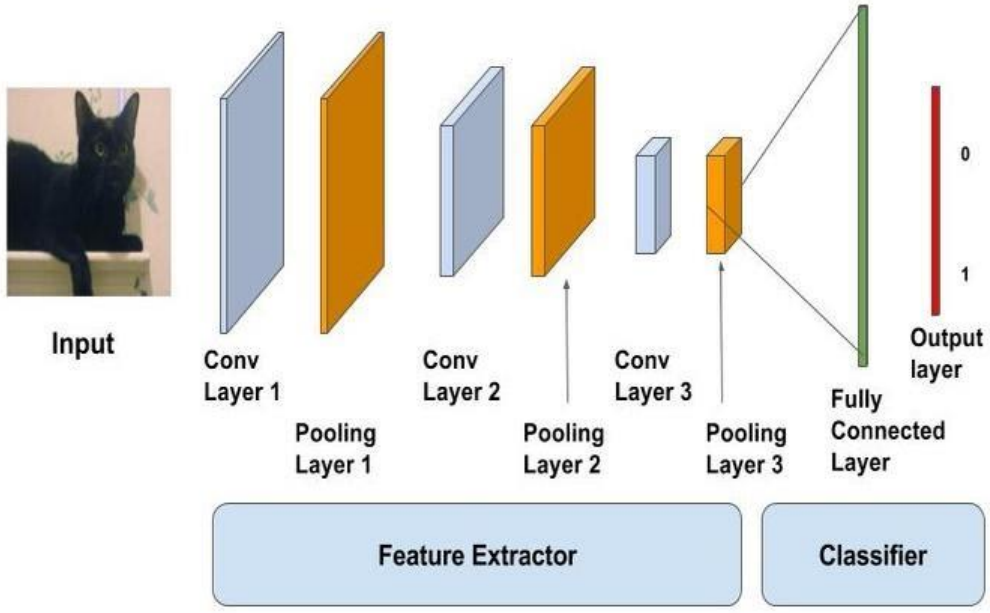

Figure 13. A ConvNet model for image classification (dog or cat) problem using Deep Learning.

When working on a new image classification problem with a new dataset that is small, it is not necessary to build and retrain the whole network from scratch. Instead, the common and more suitable approach is to make use of the above-mentioned pre-trained models to solve the problem. This technique is called transfer learning.

All of the model layers excluding the output layer can be used as a feature extractor. With other datasets and problems, this feature extractor can also be used to generate feature vectors, based on the hypothesis that images all have certain analogous characteristics. The output layer is substituted with a new classifier

(Softmax Regression or multi-class SVM with new random weights) but with the number of units matches the number of classes of these new problems. From then, there are two ways to leverage pre-trained Deep Learning models to have a much better result compare to using hand-crafted features:

- The first approach is called “ConvNet as fixed feature extractor” where only the last (output) layer needs to be trained on the new data for the new problem while all other layers’ weights are frozen (keep the same).
- The second approach is called “fine-tuning” where the last few layers are updated (trained) for a few epochs (loops). This technique is based on the observation that the early layers in the ConvNet often extract universal features of the images such as edges or curves which are low-level features while the last layers often include unique features (high-level feature) of the datasets such as cat’s nose, car’s wheel, etc. (Github no date.)

2.7 Deep Learning-based methods for object detection

Object detection is a Computer Vision task for which the goal is to locate occurrences of objects in the images. This task contains both the classification problem (classify objects into classes) and the regression problem (localize the objects by outputting their positional information). Like image classification problem, before the age of Deep Learning, Machine Learning was commonly applied for object detection problem with handcrafted feature descriptors, e.g Aggregate channel features (Bin Y., Junjie Y, Zhen L., Stan Z. L., 2014) and Viola-Jones algorithm (Paula V., Michael J., 2001). However, most of the current state-of-the-art object detectors use Deep Learning as the backbone to be able to extract rich features from input images.

Deep Learning-based object detectors can be categorized into two types with both pros and cons:

- Two-stage detector: The purpose of the first stage is to propose candidate object bounding boxes. At the second stage, features are extracted from each candidate box by Deep Learning layers, for the subsequent classification and bounding box regression tasks. This kind of detector such

as Faster R-CNN offers very high accuracies for both localization and classification but is often slower than the second type.

- One-stage detector: this approach proposes predefined boxes (anchor boxes) throughout the whole input image directly without region proposal step and then produce the predictions for these anchor boxes. This kind of detector such as YOLO and SSD gives high inference speed but the accuracy is not as good as a two-stage detector, particularly for small objects. (Licheng J.,*vet al*, 2019.)

The fundamental metric that is used for evaluating the accuracy of an object detector on a certain dataset is Intersection over Union (IoU). It is defined as the area of the intersection divided by the area of the union of a predicted box and a ground-truth box. This leads to a value between 0 and 1, which represents the overlapping degree between two boxes. An IoU score > 0.5 is considered a good prediction and the nearer to 1 the IoU score is, the better the prediction is.

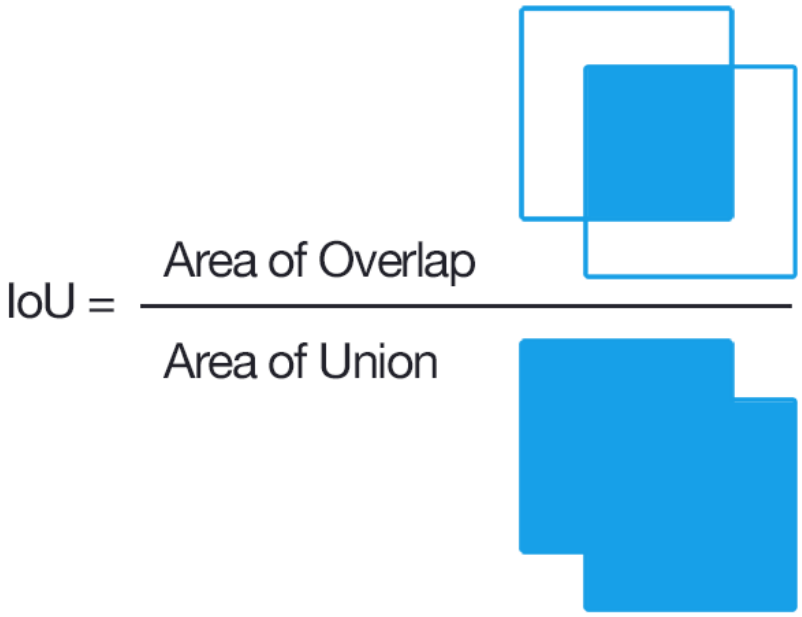

Figure 14. Illustration of the Intersection over Union (IoU) metric

2.8 Deep Learning-based methods for optical character recognition

“Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).” (Wikipedia). OCR is a well-known Computer Vision task that has many real-world applications, e.g. receipt recognition, automatic license plate recognition, or passport recognition, etc.

OCR often has two most essential steps: text detection and text recognition. Text detection is itself a special form of object detection and in some cases, can be solved by normal object detectors presented in section 2.7. This step tries to locate candidate text regions in the input image, which provides accurate and neat text instance images for the text recognition step where the cropped text instance image is converted into a string sequence. (Xiaoxue C., Lianwen J. & Yuanzhi Z., Canjie L. & Tianwei W., 2021). Traditional methods for both these steps rely on hand-crafted features such as Canny detector (Hojin C., Myungchul S., Bongjin J., 2016) or Stroke Width Transform that gave limited accuracy (particularly when the image quality is low or the background is noisy). Most recent researches like EAST (Xinyu Z., Cong Y., He W., Yuzhi W., Shuchang Z., Weiran H. & Jiajun L., 2017) or Attention OCR (Zbigniew W. et *al*., 2017) leveraged Deep Learning that presented much more superior results and advantages.

Deep Learning-based text recognition methods can be divided into two main types:

- Segmentation-based methods: This approach often includes three steps: image preprocessing, character segmentation, and character recognition. In other words, it attempts to localize/crop each character in the input text instance image, then use an image classifier like in section 2.6 to recognize each character, then finally group all recognition results into the right format (lines or block of text). Although beat traditional Machine Learning methods

using handcrafted features, this approach still has limitations since its accurateness depends greatly on character segmentation, which is a very tough problem. Moreover, it cannot capture circumstantial features/information since each character is recognized separately.

- Segmentation-free methods: This approach avoids the character segmentation/detection part by trying to recognize directly the complete text word or line. The typical pipeline of this approach includes four steps: image preprocessing, feature extraction, sequence modeling, and prediction. The image preprocessing methods improve the image quality by removing noises or rectifying. The feature extraction step generates a representation (e.g. CNNs combined with attention mechanism) containing rich features that support character recognition and eliminate inappropriate features like font and text color. The sequence modeling step (e.g. BiLSTM or LSTM) captures the circumstantial information in a series of characters, that is to say, the prediction of a character contributes to the predictions of next characters. The final prediction step aims to predict the output string sequence for the input text instance image, based on the features and information from the prior steps. (Xiaoxue C., Lianwen J., Yuanzhi Z., Canjie L. & Tianwei W. 2020.)

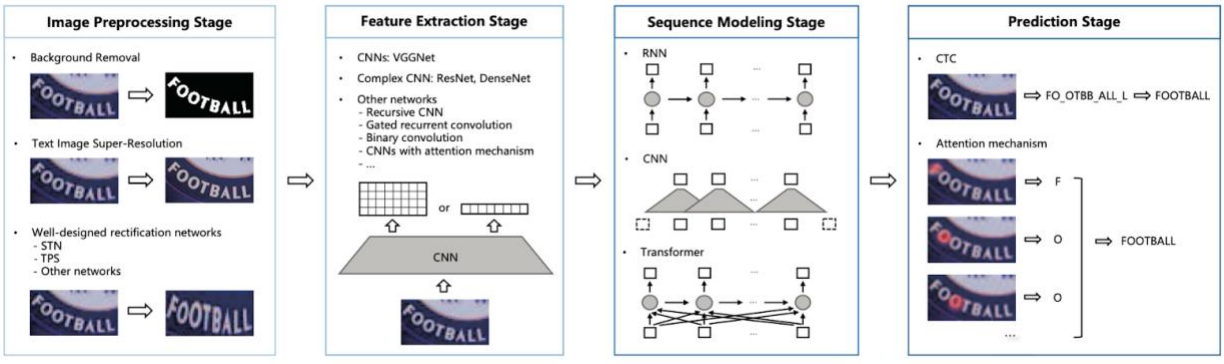

Figure 15. Example of segmentation-free methods for recognizing text (Xiaoxue C., Lianwen J., Yuanzhi Z., Canjie L. & Tianwei W. 2020)

3 PRACTICAL WORK

3.1 Solution Pipeline

The proposed solution for recognizing the Vietnamese ID card includes five main modules: Card Detector, Card Rotator, Text Detector, Text Classifier, Text Recognizer. Each module has its own models (algorithms) and each model also has its own training, evaluating, and testing process. That is to say, it is handy to improve or substitute one or some of these models to improve the entire system, which also makes the controlling and debugging process easier than an end-to-end model. Figure 16 illustrates the pipeline for the Vietnamese ID card recognition solution.

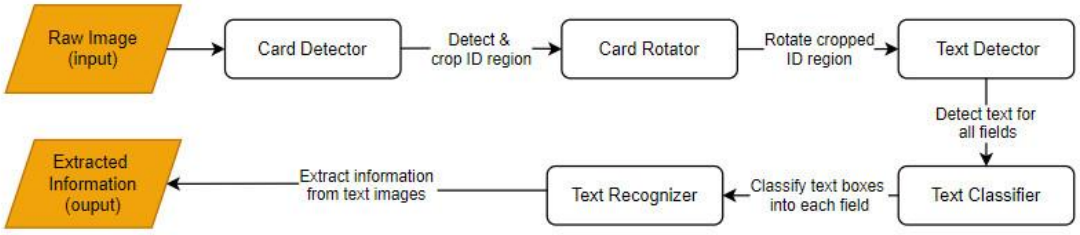

Figure 16. Pipeline for the Vietnamese ID card recognition solution

In order to build these Machine Learning/Deep Learning models, a total of 3000 Vietnamese ID card images is collected from two public sources: Google Images and Facebook groups where people upload their lost ID images, seek for help to find them. This data collecting work can be done manually or faster by tons of web crawling tools that use Python to automatically login and download images.

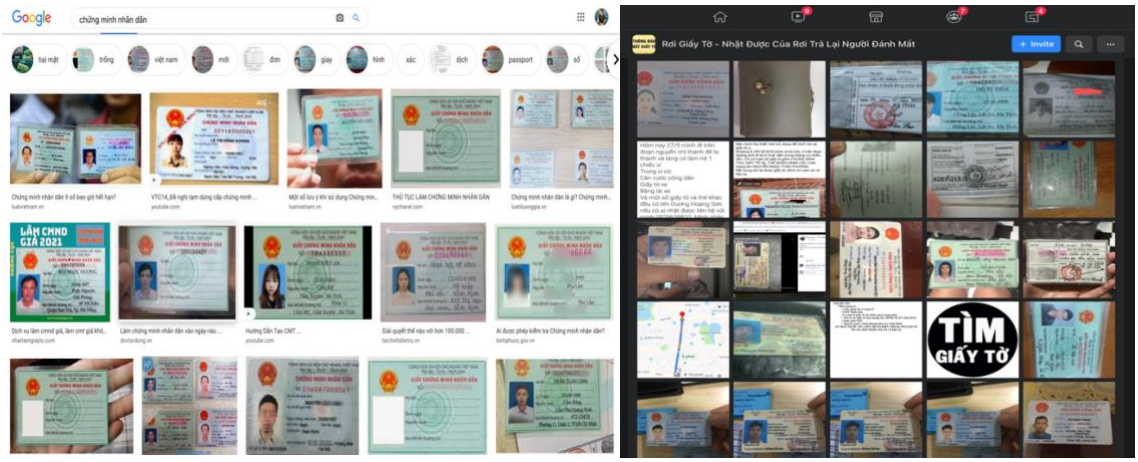

Figure 17. Examples of public sources to collect Vietnamese ID card images

These 3000 images are then randomly split into 2000 images for train phase and 1000 images for test phase.

3.1.1 Module 1: Card Detector

Firstly, the raw captured image is fed into the Card Detector module that detects four corners of the ID card. From these four points (corners), a quadrilateral-shape region is cropped from the original image before a perspective transformation is applied to that region to transform it into a rectangle (Figure 17).

Figure 18. Input (left) and output (right) of the Card Detector module

Object detection algorithms often just provide the output as a rectangle with its coordinates (xmin, ymin, xmax, ymax) while the input image of Card Detector model contains the ID cards with various shapes and tilt angles. So, in order to nicely crop the ID card region out of the original image, a possible approach is to consider each corner of the ID card as the target object, as shown in figure 18.

Figure 19. Illustration of how object detection model is used to crop out the ID card region

Implementing from scratch any deep learning models that give state-of-the-art results is a time-consuming and challenging task. That is the reason why TensorFlow Object Detection API is used to create reliable models quickly and with ease. A TensorFlow 2 Object Detection API tutorial (Github no date) provided by TensorFlow, gives a step-by-step guide to set up and train a custom object detector that can make use of the pre-trained weights on the COCO dataset that is a large-scale benchmark dataset for some common Computer Vision tasks including object detection. This transfer learning technique is similar to that one of image classification problem presented in section 2.6. It helps to train a new object detection model on a new dataset without the need for a large number of labeled training samples, which is very time-consuming and costly (especially for this kind of problem where each input image contains many bounding boxes of target objects).

The pre-trained model selected was EfficientDet D0, the smallest model in the new family of scalable and efficient object detectors called EfficientDet which is One-stage detectors but achieved both state-of-the-art accuracy on the COCO dataset

and better efficiency with optimized network architecture than prior models (Mingxing T., Ruoming P., Quoc V. Le, 2020).

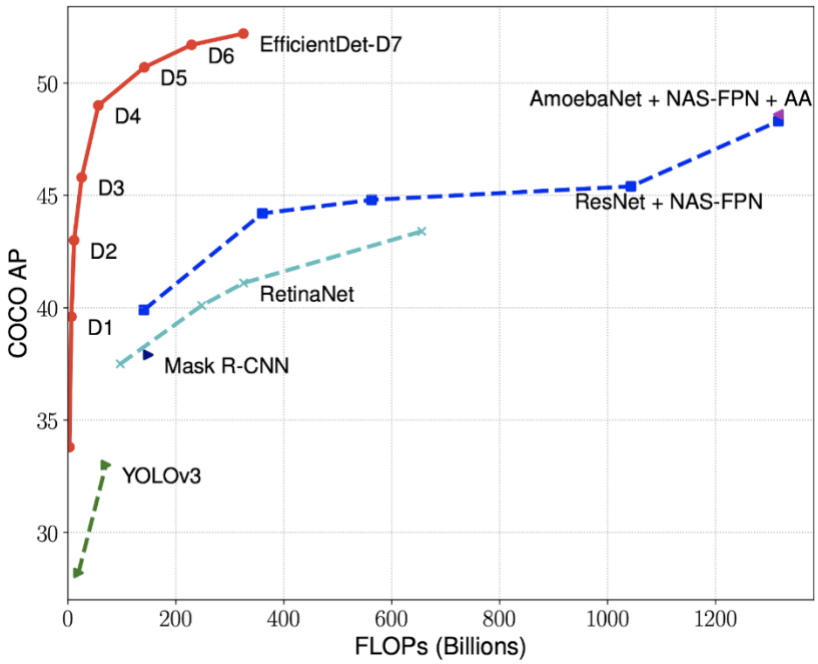

Figure 20. EfficientDet achieved state-of-the-art results on COCO dataset with much fewer parameters

Besides the labeling work which is done by using the LabelImg tool (Github no date), another important and demanding job that determines the performance of the final model is configuring the training pipeline.config file provided by the TensorFlow 2 Object Detection API. Parameters that need to be changed include num_classes (the number of classes of objects, in this case, was set to only 1), anchor_generator (anchor_scale and list of aspect_ratios), batch_size (depends on one's computing power), data_augmentation_options (to avoid overfitting), iou_threshold (post_processing parameter that can be changed after the training; depends on the overlapping of bounding boxes of the same class for each problem), etc. (Figure 20).

```
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 3.0
    aspect_ratios: 0.3
    aspect_ratios: 0.6
    aspect_ratios: 1.0
    aspect_ratios: 1.5
    aspect_ratios: 2.0
    aspect_ratios: 3.0
    aspect_ratios: 3.5
    scales_per_octave: 3
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.25
    max_detections_per_class: 100
    max_total_detections: 200
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid_focal {
      gamma: 1.5
      alpha: 0.25
```

Figure 21. A part of the pipeline.config file that needs to be configured

The model was trained on 400 labeled images (selected from 2000 training images), for maximally 300,000 iterations with the batch size of 8 that is the number of training samples fed to the model at each update (iteration) during the training loop. The model was saved to different checkpoints after every 1,000 iterations and the best checkpoint was chosen based mainly on the training loss and its performance on a validation dataset. This evaluation process was done manually by visualizing the predicting results (bounding boxes) of the selected checkpoints

since the labeling work of this problem was too costly and all the labeled data was used for training the model.

3.1.2 Module 2: Card Rotator

The output from the Card Detector module then becomes the input of the Card Rotator module that is responsible for rotating the cropped rectangle region into the right angle (Figure 21).

Figure 22. Input (left) and output (right) of the Card Rotator module

In order to do this task, two steps need to be completed. The first step is to identify the angle of the input image, i.e., 0 degree (already correctly rotated), 90 degrees anti-clockwise rotated, 270 degrees anti-clockwise rotated, or upside down (180 degrees rotated). At the second step, this identified angle is passed into a rotation function to rotate the input image into 0 degree. This second step is quite straightforward since the rotation functions are already implemented by some Computer Vision libraries, e.g imutils.rotate_bound() or cv2.rotate().

The task of the first step was solved by considering it as an image classification problem that had the input image as the cropped rectangle region of the ID card and had four output classes [0, 90, 180, 270] presenting four possible angles as mentioned above. By inspecting the data samples, this problem was evaluated as simple for modern Deep Learning techniques since the visual distinctions among the four classes are very clear. Since then, the transfer learning method (presented

in chapter 2.6) was used by fine-tuning a lightweight pre-trained model called MobileNetV2 (Mingxing T., Quoc V. L. 2019). This is a convolutional neural network architecture for which the authors' purpose was to create a model that provides both high accuracy and fast speed when running on mobile devices.

The data was collected and prepared by using the trained Card Detector module in section 3.2.1 to crop the rectangle ID card regions out of 2,000 raw training images and store them to disks. These 2,000 cropped images were then rotated manually to 0 degree before being split randomly into training and validation sets with the ratio of 9:1. From each 0-degree image of these sets, four images were generated by rotating the original image four possible angles corresponding to four classes of the image classification problem. In other words, a total of 2,000 * 4 = 8,000 images was generated, labeled with the rotated angles, and divided into training and validation sets.

The training process began by loading the MobileNetV2 model with pre-trained weights on ImageNet dataset from the Keras implementation as in Figure 22. However, since the transfer learning approach was applied, an option 'include_top=False' was passed to remove the last fully-connected layer at the top of the network.

```
net = applications.mobilenet_v2.MobileNetV2(include_top=False, pooling='avg',
                                weights='imagenet', input_shape = (224,224,3))
```

Figure 23. Loading the pre-trained MobileNetV2 model from Keras

The top of the network was then rebuilt by adding a new Softmax Regressor with four output units which is the number of classes of the solving problem. This model was intended for fine-tuning on the new dataset, so only the last 60 layers were trained while the weights of the rest were kept the same. The built model was then compiled with "categorical_crossentropy" loss (TensorFlow no date) that is often used for a multi-class classification problem, Adam optimizer (Diederik P. K., Jimmy B., 2014), and "accuracy" metric as in Figure 23.

```
model = Sequential()
model.add(net)
model.add(Dense(4, activation='softmax'))

for layer in net.layers[:-60]:
   layer.trainable = False

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
print("Compiling model done!")
```

Figure 24. Rebuild the top of MobileNetV2 model and compile it

The training and validation data was then loaded using the Keras ImageDataGenerator class that also provides image data augmentation options. Data augmentation is a technique used to artificially expand the size of a training set by adding slightly modified copies of existing data, which helps improve model performance and avoid overfitting. (Jason B., 2019). The image data augmentation techniques applied to the training set of this problem were rescaling, zooming, shifting, and shearing.

The model was then fit (trained) for maximally 20 epochs since it was previously trained quite well on the very large ImageNet dataset. The Keras EarlyStopping class (Keras no date) was used to break the training loop when the validation accuracy stopped improving for five epochs. Finally, after being trained and saved, the model was evaluated on the test set that had not been seen at the training phase, which gave a fair conclusion on the performance of the model.

3.1.3 Module 3: Text Detector

After being correctly rotated, the cropped ID card region is then fed into the Text Detector module to detect all needed text regions by outputting their bounding boxes in the form of coordinates (x_{min}, y_{min}, x_{max}, y_{max}). Since the input image is fairly rotated and straight, the object detection model was used by directly considering each text region, the QR code, and the machine-readable zone as target objects, as in Figure 24.

The pre-trained EfficientDet D1 model was selected instead of the EfficientDet D0 since this text detection problem is far more complicated and difficult than the card detection problem. The training and evaluating processes were the same as for the

card detection problem but with 1000 labeled training samples (also selected from 2000 training images), three label classes [“text”, “qrcode”, “mrz”], and some parameters (e.g anchor_ratios, data_augmentation_options) in the pipeline.config file also needed to be changed to adapt to this problem where the objects are very different.

Figure 25. Text bounding boxed detected by Text Detector module

During the training process, Tensorboard, a visualization toolkit of Tensorflow, was used to visualize the model performance on the training set by plotting the losses (Figure 25).

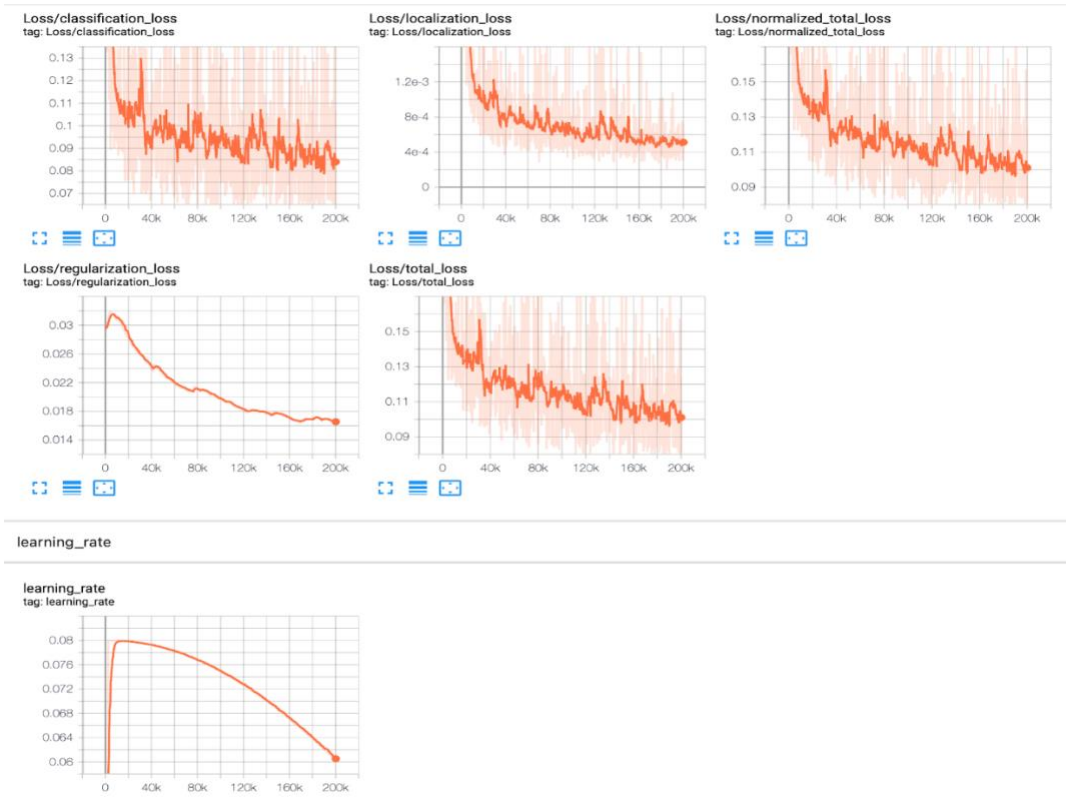

Figure 26. Visualizing model performance by using Tensorboard

3.1.4 Module 4: Text Classifier

At this point, all the needed text regions are detected, however, the information about which text region belongs to which field is not available and the only classification are “text”, “qrcode”, or “mrz” (machine-readable zone). That is when the Text Classifier module comes in. It takes the input as bounding box coordinates of text regions from the Text Detector module and classifies the field label for each text region, e.g. “type”, “id_number”, “name”, “gender”, “home”, etc. (Figure 26).

Figure 27. Text bounding box labels classified by the Text Classifier module (Class names are displayed on top of each box.)

In detail, as shown in Figure 26, the Text Detector module only outputs the detected bounding boxes for all the text regions (from that information need to be extracted) and their “unclear” class labels (“qrcode” and “text” for the front side; “text” and “mrz” for the back side). Text Detector can classify text bounding boxes into these labels since visual distinctions among these labels are very clear. However, it is very difficult to classify the bounding boxes of class label “text” further into specific fields (i.e., “name”, “gender”, “home”, etc.) because the number of training data samples for Text Detector’s model needs to be large enough (very costly in labeling work) to be able to learn the positional features that play the most important role for that classification problem. For example, the Text Detector cannot distinguish well between the text in fields “Date of birth” and “Date of expiry” without the information about where these texts are in the ID card, i.e. visually they are just both text with date format. That’s why in the solution pipeline, the Text Classifier module is proposed.

The Machine Learning model for this module is Random Forest implemented by the Scikit-learn library. The training, evaluation, and testing data containing text bounding coordinates and their labels are annotated by LabelImg tool and stored as XML files in PASCAL VOC format. For each input text bounding box, a list of features about the positional information of the input text bounding box itself and the relation to surrounding elements are generated. These features include information about where the input text bounding box is on the ID card, index of the line where it lies on, its distances to the previous and next boxes, its distances to the previous and next lines, etc. The model receives these features as input data and outputs the corresponding class labels. The best hyperparameters for the Random Forest algorithm such as n_estimators, max_depth, min_samples_split, etc. are searched by a method called Bayesian Optimization instead of Grid Search which is costly and inefficient. The model is then trained with these hyperparameters and saved to Python's Picke format for later predicting usages.

Figure 28. Example of a part of an XML file containing the text bounding coordinates and class labels

```
@staticmethod
def get_manual_feature(bbs, w, h):
    bbs = [x for x in bbs]
    didx = {tuple(x): idx for idx, x in enumerate(bbs)}
    rbbs = TextClassifier.row_bbs(bbs)
    bbs = [x for y in rbbs for x in y]
    features = []
    for idx, rbb in enumerate(rbbs):
        for i2, bb in enumerate(rbb):
            l, t, r, b = bb
            feature = [l/w, t/h, r/w, b/h, (r-l)/w, (b-t)/h, idx + 1, len(rbbs) - idx,
                       i2 + 1, len(rbb) - i2, len(rbb), len(rbbs)]
            # Distance to prev box
            if i2 == 0:
                feature.append(1)
            else:
                pl, pt, pr, pb = rbb[i2-1]
                feature.append(max(0, (l-pr)/w))
            # Distance to next box
            if i2 == len(rbb) - 1:
                feature.append(1)
            else:
                nl, nt, nr, nb = rbb[i2+1]
                feature.append(max(0, (nl-r)/w))
            # Distance to previous line
            if idx == 0:
                feature.append(1)
            else:
                pbbs = np.array(rbbs[idx-1])
                mean_b = np.mean(pbbs[:,3])
                feature.append(max(0, (t - mean_b)/h))
            # Distance to next line
            if idx == len(rbbs) - 1:
                feature.append(1)
            else:
                pbbs = np.array(rbbs[idx+1])
                mean_t = np.mean(pbbs[:,1])
                feature.append(max(0, (mean_t - b)/h))
            # Length of previous line
            if idx == 0:
                feature.append(0)
            else:
                feature.append(len(rbbs[idx-1]))
            # Length of next line
            if idx == len(rbbs) - 1:
                feature.append(0)
            else:
                feature.append(len(rbbs[idx+1]))
            #Length of 2 previous line
            if idx in [0, 1]:
                feature.append(0)
            else:
                feature.append(len(rbbs[idx-2]))
            #Length of 2 next line
            if idx in [len(rbbs) - 1, len(rbbs) - 2]:
                feature.append(0)
            else:
                feature.append(len(rbbs[idx+2]))

            features.append(feature)
    ordered_features = [None for i in range(len(bbs))]
    for idx, bb in enumerate(bbs):
        ordered_features[didx[tuple(bb)]] = features[idx]
    return np.array(ordered_features)
```

Figure 29. Source code of the function get_manual_feature() that generates features for the Random Forest model

```
#Bayesian optimization
def bayesian_optimization(function, parameters):
    n_iterations = 30
    BO = BayesianOptimization(function, parameters)
    BO.maximize(n_iter=n_iterations)

    return BO.max

#Define function to optimize and its hyperparameters
#random forest classifier
def rfc_optimization(cv_splits, X_train, y_train):
    def function(n_estimators, max_depth, min_samples_split, min_samples_leaf):
        return cross_val_score(
            RandomForestClassifier(
                n_estimators=int(max(n_estimators,0)),
                max_depth=int(max(max_depth,1)),
                min_samples_split=int(max(min_samples_split,2)),
                min_samples_leaf=int(max(min_samples_leaf,1)),
                max_features=1,
                n_jobs=-1,
                random_state=42,
                class_weight="balanced"),
            X=X_train,
            y=y_train,
            cv=cv_splits,
            scoring="f1_weighted",
            n_jobs=-1).mean()

    parameters = {"n_estimators": (50, 200),
                "max_depth": (10, 40),
                "min_samples_split": (2, 10),
                "min_samples_leaf": (1, 8),
                }

    return function, parameters
```

Figure 30. Source code of implementing the Bayesian optimization for Random Forest.

3.1.5 Module 5: Text Recognizer

Next, classified text instance images are cropped from the ID card region (output of the Card Rotator module) and fed into the Text Recognizer module that is responsible for converting them into strings and reorganizing all output information into the right format (Figure 30).

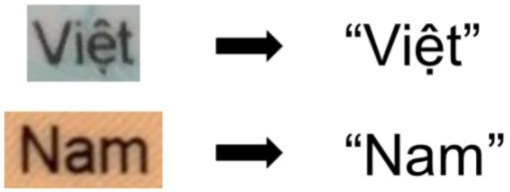

Figure 31. Input (left) and output (right) of the Text Recognizer module

The method used was to reimplement the Attention OCR model presented in the paper Attention-based Extraction of Structured Information from Street View Imagery (Zbigniew W. et *al*., 2017) and train it on Vietnamese ID card data. This model is also a deep neural network based on ConvNets, RNNs, and a novel attention mechanism that outperformed the previous state-of-the-art model Smith'16 (84.2% accuracy compared to 72.46%) on the challenging French Street Name Signs (FSNS) dataset.

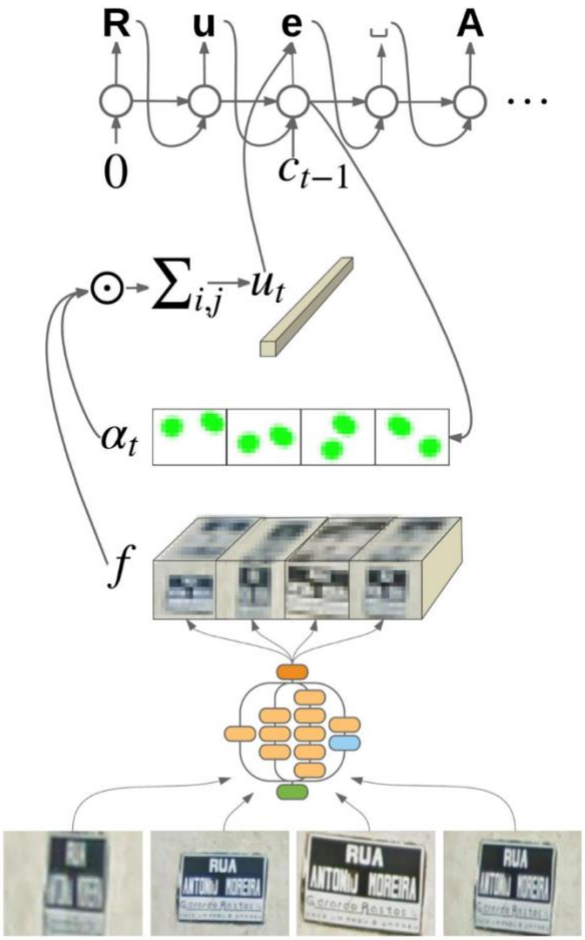

Figure 32. Architecture of the Attention OCR model in the original paper

This Attention OCR model follows the segmentation-free method design as presented in Section 2.8. The feature representation/extractor is a combination of a convolutional neural network (CNN) and a novel attention mechanism. When re-implementing the model, the pre-trained EfficientNetB1 was used instead of the three CNN models mentioned in the paper since EfficientNet was the new state-of-the-art model on the ImageNet dataset while being much lighter. The sequence model used is an RNN, precisely, a long short-term memory (LSTM) network (Sepp H. & Jürgen S., 1997).

For each text field (e.g. id_number, name, or dob), a separate model was trained on a separate dataset, which provided higher accuracies than having an all-in-one recognition model for all fields. Although there are only 2000 raw images collected for training, the final training data used was quite large because synthetic data are generated by tool https://github.com/Belval/TextRecognitionDataGenerator and combined with the original data. For example, field “doi” (date of issue) had 900,000 text instance images after further applying image data augmentation techniques like randomly rotating, changing brightness, changing contrast, adding noises, etc. The model was trained for maximally 30 epochs with a batch size of 64, the optimizer was Adam with a learning rate of 10e-4.

3.2 Tools

All the work done in the practical part uses Python, which is considered the best programming language for developing Machine Learning applications.

The main Python libraries and frameworks used while working on this thesis are: TensorFlow (for building Deep Learning models), Scikit-learn (for training traditional Machine Learning algorithms), Numpy (for scientific computation), Matplotlib (for data visualization), OpenCV (for real-time computer vision), etc.

Training the Deep Learning models such as object detection and Attention OCR means working mainly with matrix calculations, which needs GPUs to make it much faster by carrying out these computations in parallel. A cloud server with GeForce RTX 2080 Ti GPU was used, which was a powerful GPU but still took 2-4 days of

training for each Deep Learning model (it would have taken weeks or months of training if just CPU had been available).

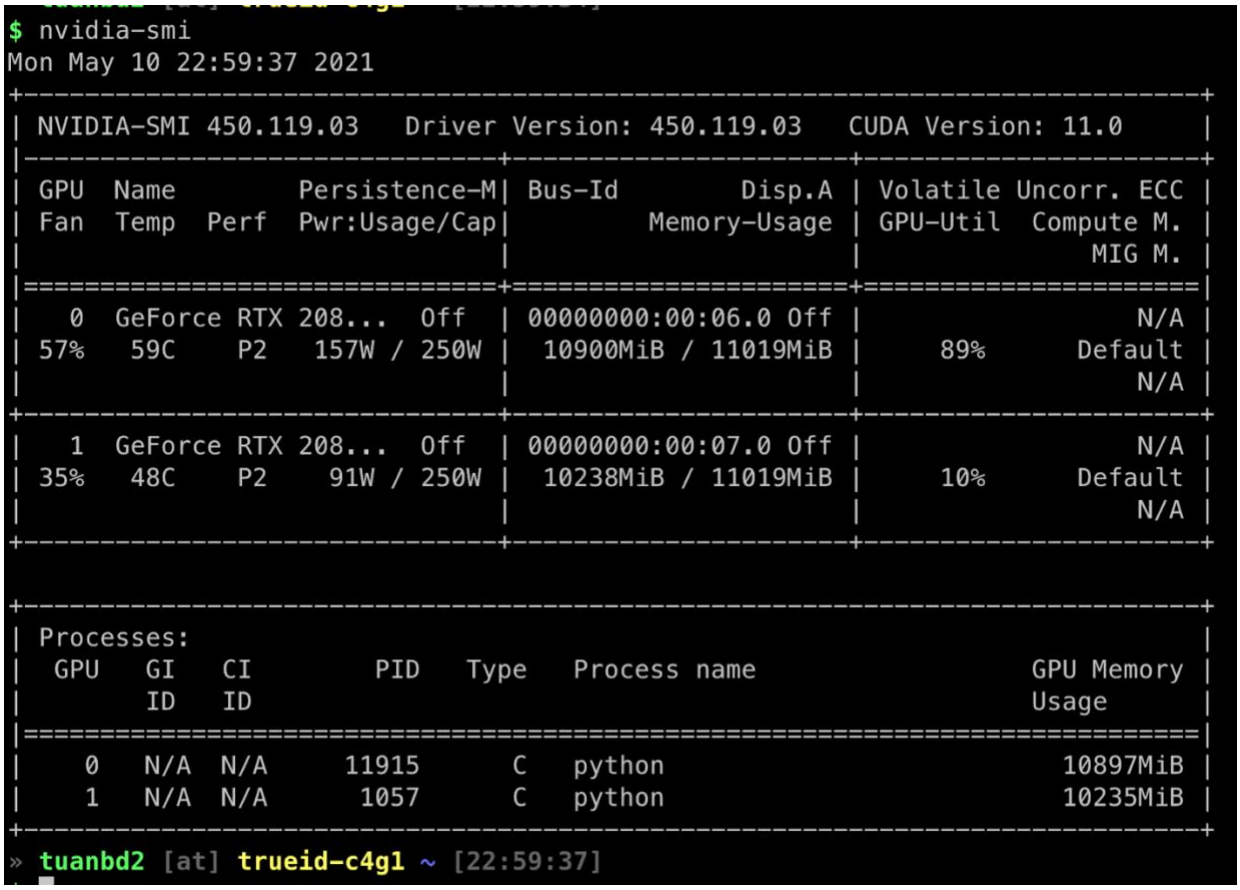

Figure 33. Information about the GPUs used

4 RESULTS

For each Machine Learning or Deep Learning model in the pipeline, after being trained/fit on the training set and evaluated/tuned on the validation set, it was assessed one more time on the test set that was held out and unseen by the model before to obtain a final unbiased evaluation on the performance of each model. Table 1 illustrates the test accuracies of these models.

Model	Test Set size (mages)	Test Accuracy
Card Detector	1000	99.7%
Card Rotator	4000	100%
Text Detector	1000	98.3%
Text Classifier	1000	100%

Table 1. Test accuracies of the separate models

Since the Text Recognizer module includes models for each text field, test accuracies were also obtained for each of them (Table 2).

Field	Test Set size (text instance image)	Test Accuracy
ID Number	1000	100%
Name	1000	97.7%
Date of Birth	1000	99.8%
Date of Expiry	1000	98.6%
Residence	1000	95.4%
Date of issue	1000	98.8%
Place of issue	1000	99.3%

Table 2. Test accuracies of Text Recognition models for some important fields

All the above models and modules were finally combined together to create the IDCardReader class that receives the input as an image capturing the Vietnamese ID card (input of Card Detector module) and returns all the extracted information for all fields and the corresponding confidence scores output by the Text Recognizer module.

However, each model was just tested separately while in reality, the performances of the prior models have very big impacts on the following ones. That is the reason why a final test was taken on the full pipeline (the IDCardReader class) and the results were promising and expectable (Table 3).

Field	Test Accuracy
ID number	99.89%
Name	97.48%
Date of birth	99.80%
Date of expiry	98.47%
Residence	93.43%
Date of issue	98.50%
Place of issue	99.00%

Table 3. Final test accuracies of the full pipeline on a test set containing 1000 raw images

In terms of inference speed, the full pipeline takes a normal laptop (MacBook Pro 2020) about 0.9 second on average to extract all information from an input image.

5 CONCLUSION

Identity document verification is an important task of any KYC process but it often costs both customers' and employees' time and efforts. This problem can be solved with the rapid development of Artificial Intelligence and high technologies like identity document recognition. The goal of this thesis was to propose a solution for the identity document recognition problem using AI to make the identity document verification task more automatic in an eKYC procedure and at the same time, explain how Machine Learning and Deep Learning may be adopted to build the main modules of that solution.

Overall, the above goal was achieved since the background about KYC, eKYC, Identity document recognition, AI, Machine Learning and Deep Learning methods like random forest, deep neural network, and convolutional neural network were presented at a high level as well as how these concepts can be applied to solve some common tasks relevant to the proposed solution pipeline. The solution for the identity document recognition problem was also proposed and achieved high accuracies in main fields of the Vietnamese ID card while having a high speed (under one second on a personal laptop). That is to say, it is capable of partially

replacing the traditional identity document verification task in an eKYC procedure (incorrectly recognized cases return very low confidence scores and manual work can then correct them).

Future improvements of the proposed solution may be seen in several ways. Firstly, the accuracies of all Machine Learning and Deep Learning models used in the pipeline can be improved by increasing the quantity and quality of data since more than half of the training and validation data was labeled by the models themselves. Secondly, more recent state-of-the-art architectures can also be used for the text detection and text recognition models that need the most improvement in terms of both accuracy and efficiency. For example, Transformer OCR can be applied to replace Attention OCR while CenterNet MobileNetV2 FPN Keypoints outperforms EfficientDet D1 on the COCO dataset. Moreover, information in the QR code and Machine-readable zone of the newly chip-based ID card may be read to cross-check the existing extracted information. This also boosts the accuracy and automation of the solution much more. Last but not least, the inference speed can be even higher if powerful CPUs or GPUs are available.

REFERENCES

Alex J. S., Robert C. W., Peter L., 1999. New Support Vector Algorithms. PDF Document. Available at: https://www.stat.purdue.edu/~yuzhu/stat598m3/Papers/NewSVM.pdf

Analytics Vidhya. 2016. Tree based algorithms: A complete tutorial from Scratch (in R & Python). WWW document. Available at: https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/?utm_source=blog&utm_medium=decision-tree-vs-random-forest-algorithm#nine [Accessed 9 May 2021].

Anastasua K. & Listlink. 2020. Deep neural networks. WWW document. Available at: https://www.kdnuggets.com/2020/02/deep-neural-networks.html [Accessed 9 May 2021].

Arner D, Barberis J, Buckly R. 2016. The emergence of regtech 2.0: from know your customer to know your data. *J Financ Transform* 44: 77-86.

Bill T. & Navneet D. 2015. Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol 1, 886-893.

Bin Y., Junjie Y, Zhen L., Stan Z. L. 2014. Rapid Object Detection using a boosted cascade of simple features. *IEEE International Joint Conference on Biometrics*.

Christian. 2020. A shallow neural network for simple nonlinear classification. WWW document. Available at: https://scipython.com/blog/a-shallow-neural-network-for-simple-nonlinear-classification/ [Accessed 9 May 2021].

Common Lounge. 2018. Ensemble methods (Part 3): Meta-learning, Stacking and Mixture of Experts. WWW document. Available at: https://www.commonlounge.com/discussion/1697ade39ac142988861daff4da7f27d [Accessed 9 May 2021].

David G. L. 1999. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol 2, 1150-1157.

Diederik P. K., Jimmy B. 2014. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

FPT.AI. 2020. A comparison of traditional and online customer identification process. WWW document. Available at: https://fpt.ai/comparison-traditional-and-online-customer-identification-process [Accessed 9 May 2021].

Github. No date. Image Classification. WWW document. Available at: https://cs231n.github.io/classification/#image-classification [Accessed 9 May 2021].

Github. No date. Labellimg. WWW Document. Available at: https://github.com/tzutalin/labelImg [Accessed 9 May 2021].

Github. No date. TensorFlow 2 Object Detection API tutorial. WWW Document. Available at: https://github.com/sglvladi/TensorFlowObjectDetectionTutorial/blob/master/docs/source/index.rst [Accessed 9 May 2021].

Github. No date. Transfer Learning. WWW document. Available at: https://cs231n.github.io/transfer-learning/ [Accessed 9 May 2021].

Hacker Noon. No date. Difference between Artificial Intelligence, Machine Learning, and Deep Learning. WWW document. Available at: https://hackernoon.com/difference-between-artificial-intelligence-machine-learning-and-deep-learning-1pcv3zeg [Accessed 9 May 2021].

Hojin C., Myungchul S., Bongjin J. 2016. Canny Text Detector: Fast and Robust Scene Text Localization Algorithm. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3566-3573.

Jason B. 2019. How to Configure Image Data Augmentation in Keras. WWW Document. Available at: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ [Accessed 9 May 2021].

Keras. No date. EarlyStopping. WWW Document. Available at: https://keras.io/api/callbacks/early_stopping/ [Accessed 9 May 2021].

Licheng J.,v*et al*. 2019. A Survey of Deep Learning-Based Object Detection. *IEEE Access*. Vol. 7, 128837-128868.

Mingxing T., Ruoming P., Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 10778-10787.

Mingxing T., Quoc V. L. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning*.

Mitchell, T. 1997. Machine Learning. 1st edition. New York: McGraw-Hill, 2.
Nagesh, S. C. 2020. Decision Tree algorithm, explained. WWW document. Available at: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html [Accessed 9 May 2021].

Paperswithcode. No date. COCO (Microsoft Common Objects in Context). WWW Document. Available at: https://paperswithcode.com/dataset/coco [Accessed 9 May 2021].

Paperswithcode. No date. Image Classification on ImageNet. WWW Document. Available at: https://paperswithcode.com/sota/image-classification-on-imagenet

Paula V., Michael J. 2001. Aggregate channel features for multi-view face detection. *IEEE International Joint Conference on Biometrics*, 1-8.

Piyush M. & Samaya M. 2020. An introduction to Deep Learning. WWW document. Available at: https://developer.ibm.com/technologies/artificial-intelligence/articles/an-introduction-to-deep-learning/ [Accessed 9 May 2021].

Samuel, A. L. 1959. Some studies in Machine Learning using the game of checkers. *IBM Journal of Research and Development.* Vol. 3, no. 3, 210-229.

Satya M. 2017. Handwritten Digits Classification : An OpenCV (C++ / Python) Tutorial. WWW Document. Available at: https://learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/

Scikit Learn. No date. Decision Trees. WWW document. Available at: https://scikit-learn.org/stable/modules/tree.html [Accessed 9 May 2021].

Sepp H. & Jürgen S. 1997. Long Short-term Memory. *Neural computation. 9,* 1735-1780.

TensorFlow. No date. TensorFlow 2 Object Detection API tutorial. WWW document. Available at: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/index.html [Accessed 9 May 2021].

TensorFlow. No date. tf.keras.losses.CategoricalCrossentropy. WWW Document. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy [Accessed 9 May 2021].

Thales Group. No date. Know Your Customer in banking. WWW document. Available at: https://www.thalesgroup.com/en/markets/digital-identity-and-security/banking-payment/issuance/id-verification/know-your-customer [Accessed 9 May 2021].

Tzutalin. No date. WWW document. Available at: https://github.com/tzutalin/labelImg [Accessed 9 May 2021].

Wikipedia. No date. Bias-variance tradeoff. WWW document. Available at: https://en.wikipedia.org/wiki/Bias–variance_tradeoff [Accessed 9 May 2021].
Wikipedia. No date. Ensemble learning. WWW document. Available at: https://en.wikipedia.org/wiki/Ensemble_learning [Accessed 9 May 2021].

Xiaoxue C., Lianwen J. & Yuanzhi Z., Canjie L. & Tianwei W. 2021. Text Recognition in the Wild: A Survey. ACM Computing Surveys, 1-35.

Xinyu Z., Cong Y., He W., Yuzhi W., Shuchang Z., Weiran H. & Jiajun L.. (2017). EAST: An Efficient and Accurate Scene Text Detector.

Zbigniew W. et *al*. 2017. Attention-Based Extraction of Structured Information from Street View Imagery. *IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 844-850.

LIST OF FIGURES

LIST OF TABLES