Finding the best suited autoencoder for reducing model complexity

Kien Mai Ngoc*, Myunggwon Hwang*

Abstract

Basically, machine learning models use input data to produce results. Sometimes, the input data is too complicated for the models to learn useful patterns. Therefore, feature engineering is a crucial data preprocessing step for constructing a proper feature set to improve the performance of such models. One of the most efficient methods for automating feature engineering is the autoencoder, which transforms the data from its original space into a latent space. However certain factors, including the datasets, the machine learning models, and the number of dimensions of the latent space (denoted by *k*), should be carefully considered when using the autoencoder. In this study, we design a framework to compare two data preprocessing approaches: with and without autoencoder and to observe the impact of these factors on autoencoder. We then conduct experiments using autoencoders with classifiers on popular datasets. The empirical results provide a perspective regarding the best suited autoencoder for these factors.

Keywords : Autoencoder | feature engineering | feature extraction | feature reduction | machine learning

Ⅰ. INTRODUCTION

The performance of machine learning methods is heavily dependent on the choice of data representation (or features) in which they are applied. For that reason, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations that result in a representation of the data that can support effective machine learning [1].

The process of selecting and transforming the data is referred to as *feature engineering*. A good feature engineering measurement helps to choose the most informative features and remove irrelevant features, which results in higher accuracy and shorter processing time [2,3]. Initially, feature engineering is vital but labor−intensive [1] as it requires human experts to select and transform the data. Subsequently, many techniques have been developed to automate the process. A popular method is *feature extraction*.

The goal of feature extraction is to map the original feature space to a space with smaller or equal dimensions [2]. The reason is that sometimes, the raw data is too complicated and may contain noise or redundant information, which can make machine learning models complex but return poor results. Therefore, feature extraction attempts to find a better representation of the raw data which can reduce computation while maintaining or improving the accuracy [2].

In particular, feature extraction obtains an entirely new set of features from the pattern of the data. It can be categorized into 2 main categories: *supervised* and *unsupervised* methods. The supervised methods consider the labels and classes of data samples, whereas the unsupervised methods are based on the variation and pattern of the data [2].

Among various methods for feature reduction, this study focuses on the **autoencoder**, which makes use of neural networks to extract useful information for building classifiers or other predictors. The goal of

* University of Science and Technology (UST), 217 Gajeong−ro, Yuseong−gu, Daejeon, 34113, Korea.
* Korea Institute of Science and Technology Information (KISTI), 245 Daehak−ro, Yuseong−gu, Daejeon, 34141, Korea.
* This work was supported by Korea Institute of Science and Technology Information (KISTI).

Manuscript : 2021. 06. 23
Revised : 2021. 07. 21
Confirmation of Publication : 2021. 07. 22
Corresponding Author: Myunggwon Hwang,
e−mail: mgh@kisti.re.kr

the autoencoder is to learn a mapping from high-dimensional observations to a lower-dimensional representation space (also called latent space), such that the original observations can be reconstructed approximately from the lower-dimensional representation [3]. The lower-dimensional representation of the data is then used as the input for machine learning models rather than the original one.

Some factors should be considered when using an autoencoder to preprocess the data. These include the dataset, the models applied to the data, and the number of dimensions of latent space (denoted by k). In this study, our methodology involves designing a framework to compare two data preprocessing approaches: with and without autoencoder and to observe the impact of these factors on autoencoder. We first provide a survey of several types of autoencoders and then conducts experiments using those types for feature extraction. Concretely, we use different types of autoencoders and values of k to extract features from various datasets. Subsequently, some types of classifiers are employed to classify the extracted data into the corresponding classes. The accuracy of the classifier is used to assess which autoencoder shows the best performance on which k, which data, and which classifiers.

In the following sections, we provide an overview of the autoencoder and the classifier. We then present our methodology and experimental results. From the results acquired, we discuss which autoencoder is best suited for each dataset.

For simplicity, some notations should be defined. We denote $\boldsymbol{x}$ as the input vectors, and $\boldsymbol{z}$ as the latent code vectors which is the information extracted by the autoencoder.

II. AUTOENCODER

2.1. Base autoencoder

The concept of autoencoders has been a part of the historical landscape of neural networks for decades. Traditionally, autoencoders have been used for dimensionality reduction or feature learning. Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling [4].

In feature reduction, autoencoder, which makes use of neural networks, is an unsupervised feature extraction method. The form of an autoencoder consists of an encoder and a decoder having hidden layers. The input is fed to the encoder to produce latent code, and the output is extracted from the latent code by the decoder. A conventional autoencoder attempts to minimize the discrepancy between the input and the decoded output, or in other words, to learn an identity function. Through training, the autoencoder is expected to discover a more efficient and compressed representation of the data (presented by the latent code). Once the network is trained, the decoder part is discarded, and the output of the innermost hidden layer is used for feature extraction from the input [2].

Recent advances in autoencoder try to apply prior knowledge on the latent space to learn useful representations of the data [3]. We shall explore these types of autoencoders in the following subsection.

2.2. Denoising Autoencoder (DAE)

The traditional autoencoder faces the risk of overfitting. As the autoencoder learns the identity function, if the encoder and decoder are allowed too much capacity (in simple terms, a model's capacity is its ability to fit a wide variety of functions [4]), the autoencoder can learn to perform the copy task without extracting useful information regarding the distribution of the data.

To avoid this problem, the DAE [7,8] proposed a modification to the basic autoencoder. The input is partially corrupted by adding noise or masking it in a stochastic manner. Specifically, a fixed proportion of input dimensions are selected randomly, and their values are forced to 0. Next, the model is trained to recover the original input (not the corrupted one).

In our experiments, the DAE models have hidden layers 1000-500-250-k-250-500-1000 and the corruption proportion of 0.1 for the MNIST and Fashion MNIST datasets [5], and 0.2 for the Cifar10 dataset (the corruption proportion of 0.1, 0.2, and 0.5 give similar performance).

2.3. Variational Autoencoder (VAE)

VAEs have shown promise in generating many kinds of complicated data, including handwritten digits [9,10], faces [9,11,12], house numbers [13,14], segmentation [6], predicting the future from static images [7], and removing out-of-distribution samples [8]. The difference of the VAE is that instead of mapping the input into a fixed vector of latent space, we map it into a distribution. VAEs [9,12] aim to learn a parametric latent variable model by maximizing the marginal log-likelihood of the training data [3]. We denote the prior distribution of the latent space as $p(\boldsymbol{z})$, and the posterior distribution learned by the autoencoder model as $q_x(\boldsymbol{z})$. We also denote the distribution of the reconstruction of the input from latent code as $p(\boldsymbol{x}|\boldsymbol{z})$.

The loss function is the evidence lower bound (ELBO), which is expressed as follows:

$$ELBO = -\log p(\boldsymbol{x}|\boldsymbol{z}) + KL(q_x(\boldsymbol{z})|p(\boldsymbol{z})) \qquad (1)$$

In the first term of the loss function, we want to maximize the probability of reconstruction. In the second term, we can use the Kullback-Leibler (KL) divergence to quantify the distance between the prior and posterior distributions of the latent space, which forces the posterior distribution to be close to the prior distribution. The prior distribution is usually the standard normal distribution $p(\boldsymbol{z}) \sim N(0,1)$.

The training steps require sampling $\boldsymbol{z} \sim q_x(\boldsymbol{z})$ (because the input is reconstructed from the latent code $\boldsymbol{z}$, and not the parameters of the distribution), which is a stochastic process and can not be backpropagated. To make it trainable, the sampling can be replaced with "parameterization trick" described by Kingma et al. [9].

Our experiments use a model similar to that presented by Doersch [10]. The autoencoders have hidden layers 1000-500-250-k-250-500-1000. We also use convolutional layers in the network, which is referred to as convolutional variational autoencoder (CVAE), presented in the next subsection.

2.4. Info Variational Autoencoder (IVAE)

Kim and Mnih [11], Chen et al. [12], and Zhao et al. [13] all proposed methods regarding mutual information of x and z. According to Zhao et al. [13], the KL divergence in equation (1) has two problems: uninformative latent code and variance over-estimation in the feature space. Therefore, they use maximum mean discrepancy (MMD) [14] instead.

MMD is based on the idea that two distributions are identical if and only if all their moments are the same. Therefore, we can define a divergence by measuring the difference between the moments of two distributions $p(z)$ and $q(z)$. It can be efficiently implemented using a kernel trick [13]. Using MMD in the IVAE will maximize the mutual information between the input $\boldsymbol{x}$ and the latent code $\boldsymbol{z}$.

$$\begin{aligned} MMD\big(q(z)\,|\,p(z)\big) &= E_{p(z),p(z')}[k(z,z')] \\ &\quad - 2E_{q(z),p(z')}[k(z,z')] \qquad (2) \\ &\quad + E_{q(z),q(z')}[k(z,z')] \end{aligned}$$

where $k(z,z')$ is any universal kernel. A kernel can be intuitively interpreted as a function that measures the "similarity" of two samples. It has a large value when two samples are similar, and a small value when they are different. $MMD = 0$ if and only if $p = q$.

Our IVAE models use two convolution layers with 64 and 128 filters, respectively, one hidden layer with 1024 units, and latent space with k units for the encoder and the reverse structure for the decoder.

To assess the effect of MMD and network architecture, we also use a VAE with the same architecture as the IVAE model, while keeping its KL loss function. We refer to this model as the convolutional variational autoencoder (CVAE).

2.5. Adversarial Autoencoder (AAE)

AAEs [15] turn a standard autoencoder into a generative model by imposing a prior distribution $p(\boldsymbol{z})$ on the latent variables by penalizing some statistical divergence between $p(\boldsymbol{z})$ and $q_x(\boldsymbol{z})$ using a generative adversarial network (GAN) [3].

In addition to minimizing the reconstruction discrepancy, these autoencoders have an additional discriminator part to ensure that the encoded latent codes are similar to samples obtained from the prior distribution. As in the VAE, the prior distribution is usually the standard normal distribution. In all the experiments conducted by Makhzani et al. [15], the encoder and decoder are considered to be deterministic.

In our experiments, the autoencoder has hidden layers 1000-1000-k-1000-1000, whereas the discriminator network has layers k-1000-1000-1 (the last layer indicates whether the latent code is from the encoder or sampled from the prior distribution).

Generally, autoencoders with different architectures and loss functions can have different impacts on the extracted data. To assess these impacts, we focus on the performance of several classifiers using the extracted data. The classifiers are presented in the next section.

III. CLASSIFIERS

Several classifiers are used for experiments to magnify the effectiveness of feature extraction using autoencoder. They include Gaussian Naive Bayes, support vector machine, random forest, and neural network. We benchmark the performance of the classifiers trained on the original dataset and the extracted one.

3.1. Gaussian Naïve Bayes

Naive Bayes classifier is a simple probabilistic classifier based on the Bayes theorem. Naive Bayes considers each feature variable as an independent variable [16]. It involves the prior and posterior probability calculation of the classes in the dataset.

The prior probability of an instance x belonging to a class c is expressed as follows:

$$P(C=c)=\frac{Number\ of\ instances\ of\ class\ c}{Total\ number\ of\ instances} \quad (3)$$

The posterior probability of the occurrence of an instance x, given class c is computed as follows:

$$P(x \mid C=c)=\prod_{i=1}^{n} P(X_i=x_i \mid C=c) \quad (4)$$

where X_i is the feature variable and C is the class label.

The classification is performed by applying the Bayes theorem to calculate the probability of a particular instance $x=(x_1,\dots,x_n)$ belonging to a class.

$$\begin{aligned} &P(C=c \mid X_1=x_1,\dots,X_n=x_n)\\ &=\frac{P(C=c)*P(x \mid C=c)}{P(x)}\\ &=\frac{P(C=c)*\prod_{i=1}^{n}P(X_i=x_i \mid C=c)}{\sum_j P(C=c_j)*\prod_{i=1}^{n}P(X_i=x_i \mid C=c_j)} \end{aligned} \quad (5)$$

We can just compare the numerator for each class as the denominator is the same for all classes. The class of an instance $x=(x_1,x_2,\dots,x_n)$ is given by

$$\underset{c}{\operatorname{argmax}} P(C=c)*\prod_{i=1}^{n} P(X_i=x_i \mid C=c) \quad (6)$$

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. The training data are segmented by class, and the mean and variance of each class are calculated [16]. Therefore, the following formula can be used to estimate the probabilities of a continuous dataset.

$$P(X_i=x_i \mid C=c_j)=\frac{1}{\sqrt{2\pi\sigma_{ij}}}e^{-\frac{(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}} \quad (7)$$

where μ_{ij} and σ_{ij} are the mean and standard deviation of variable X_i for class c_j.

3.2. Support Vector Machine

Support vector machine (SVM) [17] is a classification approach. It constructs a maximum marginal hyperplane in a multidimensional space to separate different classes. There are some data points that are closest to the hyperplane and contribute to the construction of the hyperplane. They are called support vectors.

Initially, the boundary between two classes is linear. However, in practice, we are sometimes faced with non−linear class boundaries. In such cases, we could address the problem by enlarging the feature space using quadratic, cubic, and even higher−order polynomial functions of the features [18]. SVM uses a technique called the kernel trick. Here, the kernel takes a low−dimensional input space and transforms it into a higher dimensional space. In other words, the kernel trick converts a nonseparable problem to separable problems by adding more dimensions to the input space.

In this study, we use the radial basis function kernel (rbf), which has the formula:

$$K(x_i,x_i')=\exp\left(-\gamma\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2\right) \quad (8)$$

where γ is the kernel coefficient, ranging from 0 to 1. It is set to $\frac{1}{n\_features}*X.var()$ by default.

3.3. Random forest

Random forests [19] are an ensemble model made of decision trees. Several decision trees are built on bootstrapped training samples that are generated by taking repeated samples from the (single) training dataset. Each time a split in a tree is considered, a random sample of m features is chosen as split candidates from the full set of p features. The split can use only one of those m features [18]. By combining hundreds or even thousands of trees, a random forest will usually outperform the individual tree.

In classification, for a given test observation, we can record the class predicted by each of the trees and take a *majority vote*. The overall prediction is the most commonly occurring class among the predictions [18].

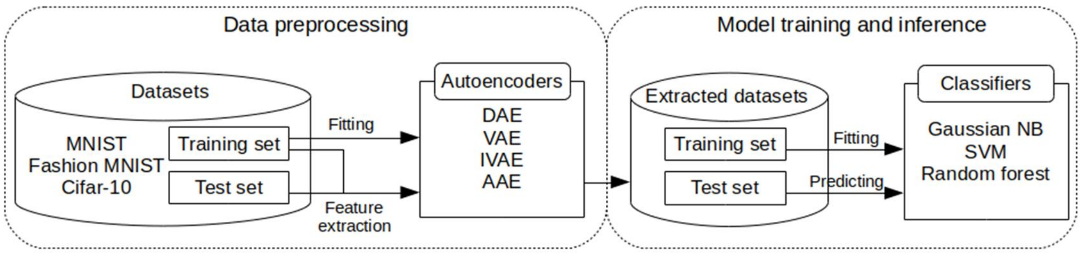


Fig. 1. Process of experiments using autoencoders in the data preprocessing step.

We use 100 classifier trees for each random forest in this study.

3.4. Neural networks

In this study, we employ a convolutional neural network as the data are images. Convolution networks combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: *local receptive fields*, *shared weights* (or weight replication), and spatial or temporal *sub-sampling* [20].

The input layer receives images that are approximately size normalized and centered. Each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer. With local receptive fields, neurons can extract elementary visual features such as oriented edges, endpoints, and corners, or similar features in other signals, such as speech spectrograms. These features are then combined by the subsequent layers to detect higher-order features. Units in a layer are organized in planes within which all units share the same set of weights. The set of outputs of the units in such a plane is called a *feature map* [20].

Once a feature has been detected, its exact location becomes less important. Only its approximate position relative to other features is relevant. A simple way to reduce the precision with which the position of distinctive features is encoded in a feature map is to reduce the spatial resolution of the feature map. This can be achieved with a so-called *sub-sampling layers* which performs a local averaging and a sub-sampling, reducing the resolution of the feature map, and reducing the sensitivity of the output to shifts and distortions [20].

The architecture used in this study follows the LeNet-5 architecture presented in the work of LeCun et al. [20]. It comprises seven layers, not counting the input layer. There are two convolutional layers (with 6 and 16 5x5-filters, respectively) followed by an average sub-sampling layer each. After that, there are one convolutional layer with 120 5x5-filters and two fully connected layers with 84 and 10 units. The activation function is rectified linear unit (ReLU) for all layers, except softmax for the output layer because we perform classification on this layer. The loss function is the sparse categorical cross-entropy.

IV. METHODOLOGY

As mentioned earlier, in this study, we use empirical results to assess the factors affecting the autoencoder, including the datasets, the machine learning models, and the number of dimensions of the latent space (denoted by k). Hence, our method involves designing a framework for conducting experiments and analyzing the acquired results.

The problem in this study is image classification. We employ popular machine learning algorithms to classify images from several datasets into their corresponding categories. The image or data can be preprocessed in advance by the autoencoders.

The goal of using autoencoder for feature extraction is to reduce the dimensions of the feature space while retaining useful information for building the classifiers. Therefore, we compared two data preprocessing approaches for training classifiers: without and with autoencoder. In the latter one, we used different combinations of autoencoders, numbers of latent dimensions, and classifiers on different datasets to provide a comprehensive comparison of these factors.

In the first approach, we employed the classifier models mentioned in Section 3 to classify the original datasets without any preprocessing steps. The performance of these classifiers in terms of accuracy and running time was considered as the baseline for further comparison.

In the second approach, we conducted experiments using autoencoders for preprocessing data. As shown in Figure 1, the overall process consists of two main steps: data preprocessing, and model training and inference. Each dataset consists of a training and test set. In the data preprocessing step, the original training set was used for training the autoencoders in an unsupervised manner. For each dataset, we ran different types of autoencoders and different numbers of latent space dimensions k. After being trained, the autoencoders transformed the original training and test datasets into extracted training and test datasets. In the model training and inference step, these two extracted datasets, including labels, were then used as the training and test set for fitting the classifiers and making predictions, respectively. The classifiers included all types mentioned in Section 3 apart from the neural network because its architecture is specified for the original data.

Subsequently, we compared the accuracy and elapsed time of the classification between the two approaches as well as among the values of the latent space dimension k, the classifiers, and the datasets in the same approach to evaluate the efficiency of the autoencoders. We also compared different network architectures and loss functions used in autoencoders. We finally provided further suggestions regarding the use of an autoencoder for feature extraction.

The main effort is not to maximize the performance of the autoencoders. We used a similar structure and hyper-parameters for each type of autoencoder to observe their performance regarding different combinations of k, classifiers, and datasets. All models were optimized by Adam optimization [21] (Kingma & Ba, 2015). All experiments were implemented using Tensorflow[1].

V. EXPERIMENTS

This section presents the experimental results. We first introduce datasets used for experiments in subsection 5.1. The baseline performances of the first approach are shown in subsection 5.2. The results of the approach using autoencoders are presented in subsection 5.3. A comparison regarding convolution and info variation autoencoders is included in subsection 5.4.

[1] The source code is available on Github at https://github.com/KienMN/Autoencoder-Experiments.

5.1. Datasets

The datasets used in this study include: MNIST [22], Fashion MNIST [23], and Cifar-10 [24].

- MNIST is a dataset of handwritten digits. It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image associated with a label of a digit.
- Fashion MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image associated with a label from 10 classes.
- The Cifar-10 dataset consists of 60,000 32x32 color images in 10 classes with 6,000 images per class. There are 50,000 training images and 10,000 test images.

Table 1 summarizes the information regarding the datasets.

Table 1. Summary of the datasets

Summary	MNIST	Fashion MNIST	Cifar-10
Training examples	60,000	60,000	50,000
Test examples	10,000	10,000	10,000
Image size	28x28	28x28	32x32
Image type	gray	Gray	color
Number of channels	1	1	3

The reason for selecting these datasets is that they are popular in the field of machine learning, they require little domain knowledge, and it is difficult to extract features manually because of their large number of dimensions. One special point is that they are all image data. Therefore, they might be suitable for some network architecture.

5.2. Baseline models

We ran the neural network and other classifiers, including Gaussian Naive Bayes, SVM, and Random Forest, on the original dataset. The accuracy of these models was considered as the baseline for the benchmark. The results are shown in Table 2.

Table 2. Classification accuracy (%) of the baseline models

Models	MNIST	Fashion MNIST	Cifar−10
Neural network	98.31	86.78	56.52
GaussianNB	55.58	58.56	29.76
SVM	97.92	88.29	54.36
Random forest	97	87.84	46.85

As shown in Table 2, the accuracy of the Gaussian NB classifiers is relatively low over all datasets. Interestingly, without data preprocessing, SVM and random forest classifiers performed nearly as well as neural network classifiers.

Table 3. Running time (s) of the baseline models

Models	MNIST	Fashion MNIST	Cifar−10
Neural network	45.47	45.36	45.47
GaussianNB	0.83	0.77	3.19
SVM	580.35	867.79	10632.66
Random forest	28.04	62.12	172.57

According to Table 3, the running time of the neural network, Gaussian NB, and random forest classifiers is significantly lower than that of SVM. The elapsed time of the neural network is likely to depend on the architecture of the network as we use similar architectures for all datasets. This may be because of the implementation and optimization of the deep learning library. The running time of other classifiers increases when the dataset is more sophisticated.

5.3. Using autoencoders

We used autoencoders for feature extraction from the original dataset before fitting the data into the classifiers. The accuracy and running time of the classification when applying the autoencoder are presented in Figures 2 − 7.

Figures 2 to 7 show the accuracy and the running time (autoencoder and classifier) of the classification on the test set of the MNIST, Fashion MNIST, and Cifar−10 datasets, respectively, depending on the number of latent space dimensions (k). The horizontal axis represents the values of k, and the vertical axis represents the accuracy or the running time of the classification. Each line indicates the metric of a model, which is a combination of an autoencoder and a classifier, on the test set when using the data extracted by the autoencoder. The same color is used for the same type of autoencoder, and the same line style is used for the same classifier. GBM, SVM, and RF are shorthand for Gaussian NB, support vector machine, and random forest. The running time of the autoencoder is not considered here because we consider autoencoder as a pre−training step.

The first observation is the amount of reduced data. The value of k is now below 30, which indicates that the size of the encoded data is approximately 1% − 3% compared to that of the original data. As a result, the complexity of the models in terms of running time and space will decrease significantly, as presented below.

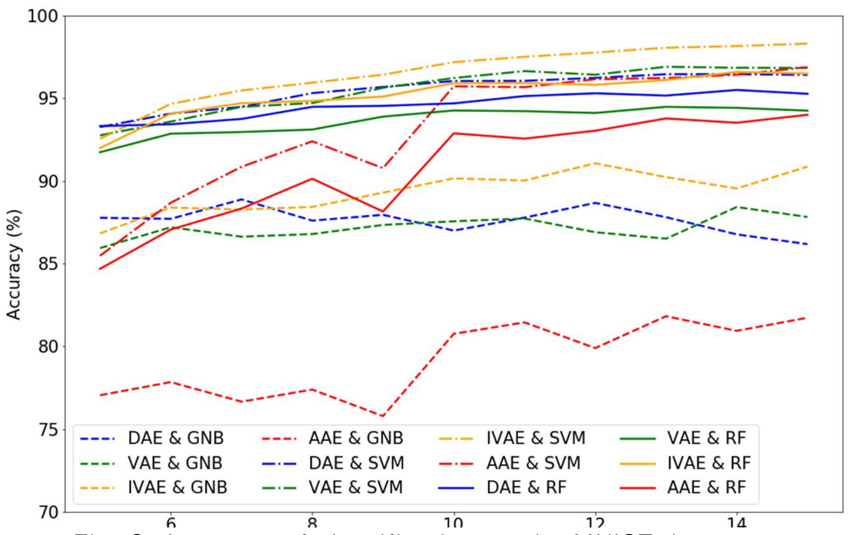


Fig. 2. Accuracy of classification on the MNIST dataset

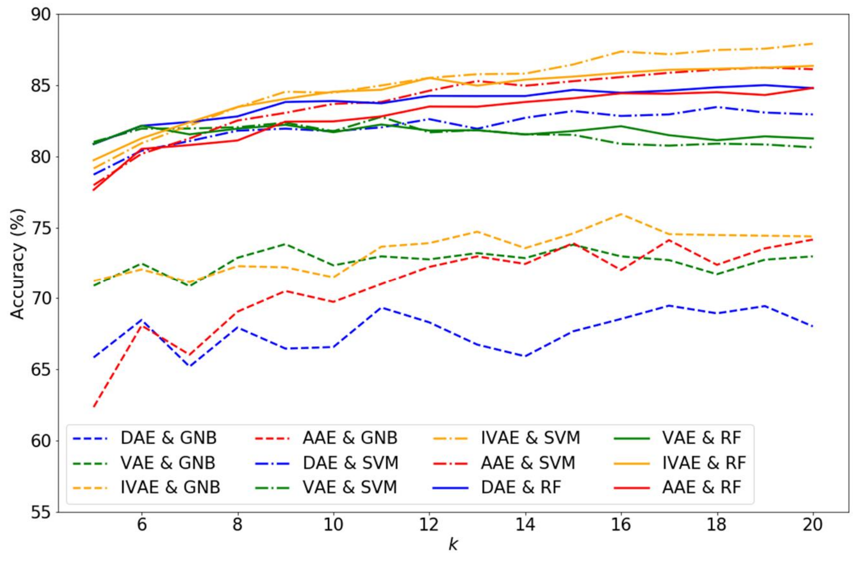


Fig. 3. Accuracy of classification on the Fashion MNIST dataset

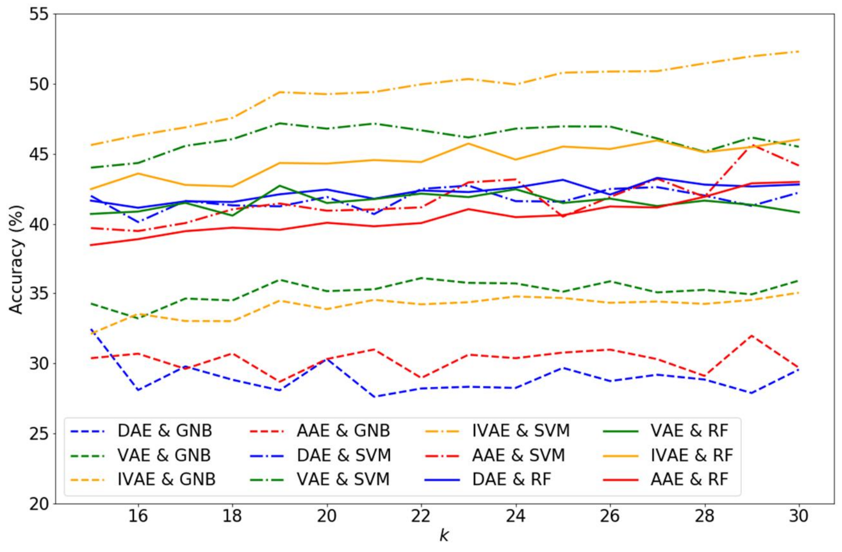


Fig. 4. Accuracy of classification on the Cifar−10 dataset

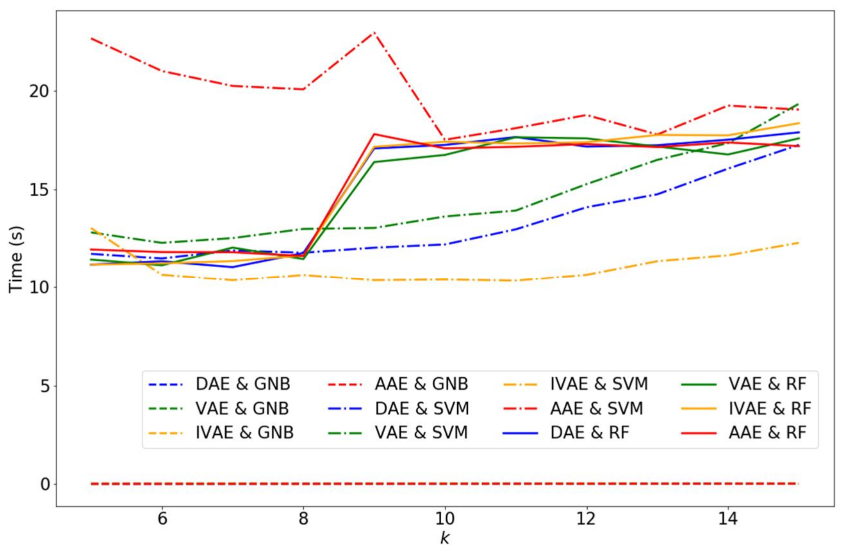


Fig. 5. Running time of classification on the MNIST dataset

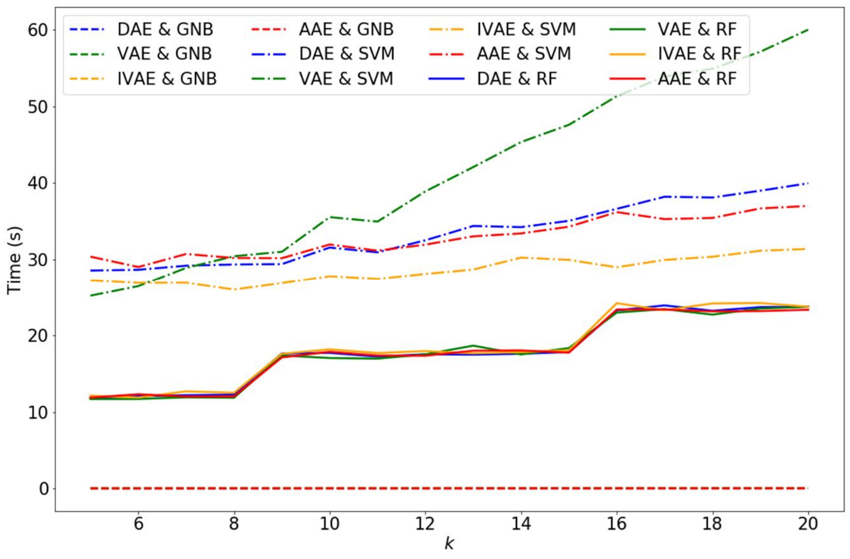


Fig. 6. Running time of classification on the Fashion MNIST dataset

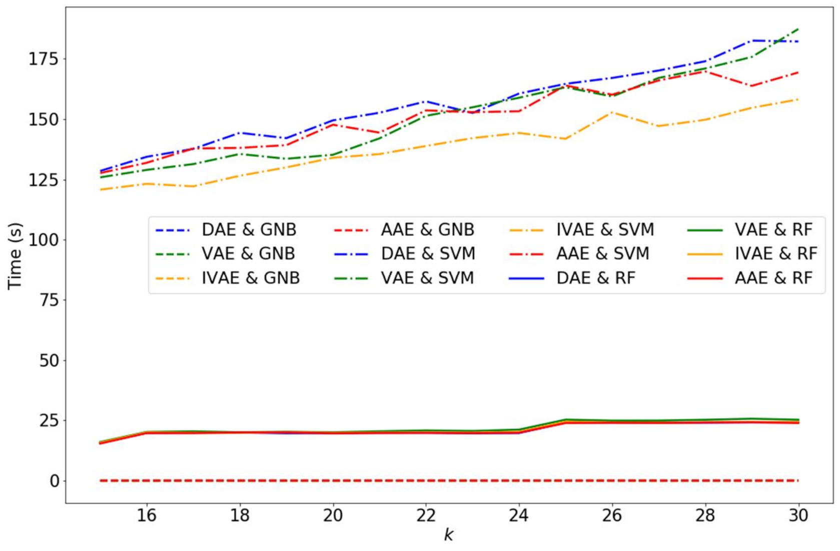


Fig. 7. Running time of classification on the Cifar−10 dataset

As shown in Figure 2, overall, the usage of the autoencoder enhances the performance of the Gaussian NB classifier from 55.58% to at least over 80% while maintaining the performance of the SVM and random forest on significantly low dimensionality data. The accuracy of the classifiers increases gently with the increase in k. The performance of the IVAE is the best among all autoencoders. Turning to the running time, as shown in Figure 5, the elapsed time of the SVM drops dramatically from 580s when running on the entire data to under 25s when running on the extracted data. One interesting thing to note is that there is a small leap in the performance of the AAE when the value of k is 10. Correspondingly, the running time of the SVM using data extracted by the AAE decreases sharply when the value of k is 10. In this case, the extracted data are well separated in the space, so that the SVM can assess fewer support vectors than before.

According to Figures 3 and 6, on the Fashion MNIST dataset, similar to the MNIST dataset, autoencoders help to improve the accuracy of the Gaussian NB classifier from 58.56% to over 70%. Using the autoencoder still helps to reduce the running time of the SVM from over 850s to under 60s and that of random forest from over 60s to under 25s at the cost of a small reduction in the accuracy (by $1-5\%$ and $3-5\%$, respectively). The increase in k gives a small boost to the result, but gradually, the effectiveness becomes insignificant. The IVAE still demonstrates the best performance.

On the Cifar−10 dataset, as shown in Figures 4 and 7, only the VAE and IVAE help to improve the accuracy of the Gaussian NB but insignificantly (approximately 5%). For the IVAE, increasing k increases the accuracy. Apart from that, increasing k has little effect on the accuracy of the classifiers. The running time of the SVM and random forest can be reduced by using extracted data from the autoencoder (from over 10,500s down to under 200s and near 180s to around 25s, respectively) at the cost of reduction in accuracy (at least for 2% both).

In all experiments, the autoencoder approach could not beat the neural network approach in terms of accuracy. The combination of the IVAE and the SVM demonstrated the best performance among the combinations. The reasons are clarified in the suggestion section.

Generally, the dataset has some impacts on the autoencoder model. In all datasets, the IVAE performed well because of the convolution layers in its architecture, which is suitable for image data. In a more sophisticated dataset (Cifar−10), the DAE and the AAE did not perform well because each image has three channels. Therefore, it is not simple to treat an image as a large input vector for a dense layer. Surprisingly, although it contained only dense layers, the VAE performed better than the DAE and the AAE on Cifar−10 dataset.

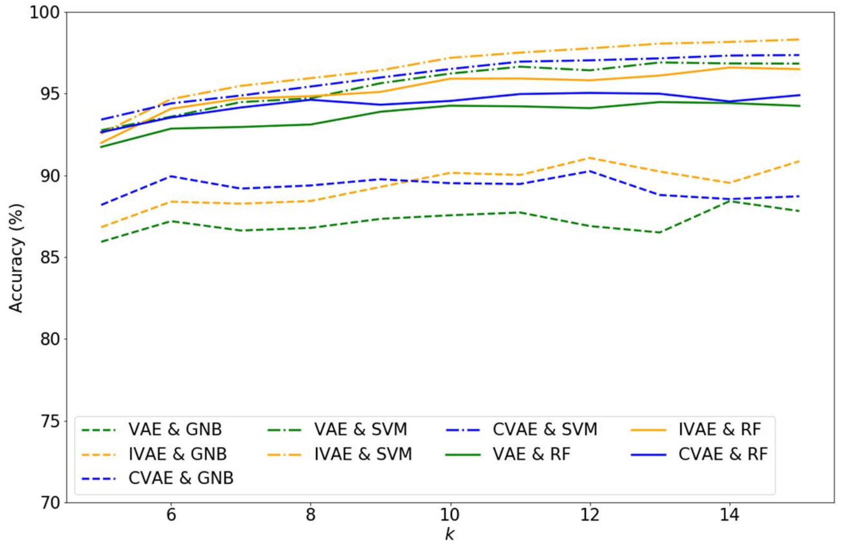


Fig. 8. Accuracy of classification on the MNIST dataset

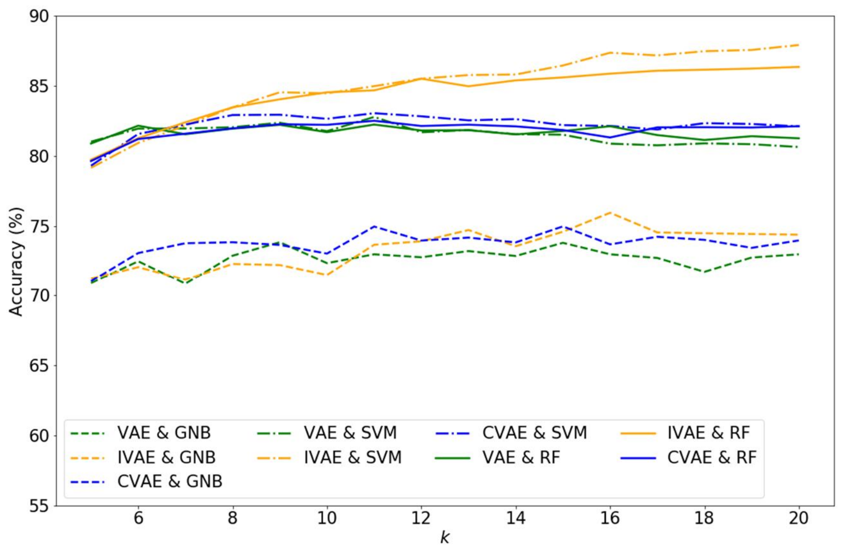


Fig. 9. Accuracy of classification on the Fashion MNIST dataset

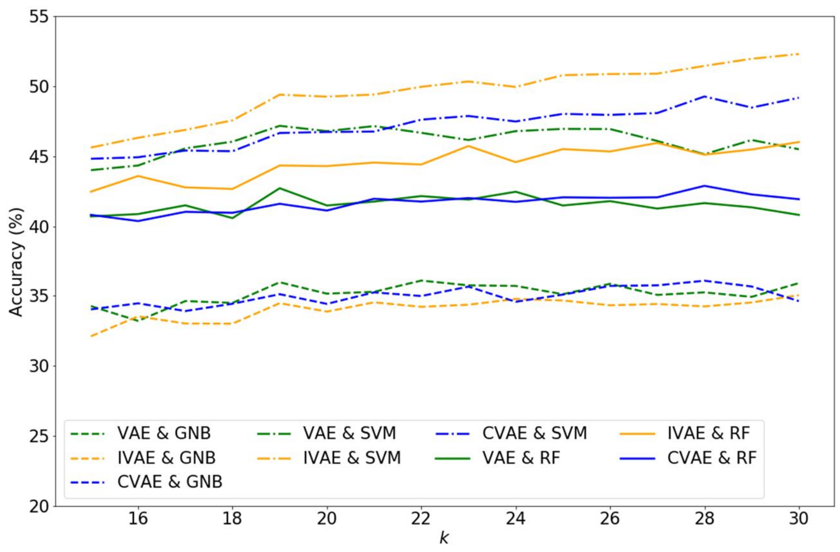


Fig. 10. Accuracy of classification on the Cifar−10 dataset

5.4. Convolution variational autoencoder versus Info variational autoencoder

As mentioned in subsection 2.4, to assess the efficiency of the network architecture and loss function, we compare the performance of three types of autoencoders. The first one is VAE, which contains only dense layers (fully connected layers) and uses the KL loss function shown in equation (1). Using the same loss function as the VAE, a Convolutional variational autoencoder (CVAE) employs additional convolutional layers that show an advantage on the image data. The last one, which is the IVAE, has the same architecture as the CVAE. However, it makes use of the MMD loss function expressed in equation (2). A summary of these three types of autoencoders is presented in Table 4, and the result is presented in Figures 8 − 10.

Table 4. Summary of the VAE, CVAE, and IVAE

Autoencoders	Network architecture	Loss function
VAE	Dense layers	KL
CVAE	Convolutional and Dense layers	KL
IVAE	Convolutional and Dense layers	MMD

Figures 8 to 10 illustrate the accuracy of classification on the data extracted by different types of autoencoders, similar to Figures 2 to 4. The autoencoders include the VAE, CVAE, and IVAE.

Overall, the performance of the IVAE is higher than that of the CVAE. In most cases, the orange line is higher than the blue one of the same line style. This indicates that its MMD loss function forces the autoencoder to encode more useful information from the input to the latent code than the KL loss function.

Meanwhile, the CVAE performs better than the VAE. The reason is that the convolutional layers allow the autoencoder to detect and extract more useful features from the image dataset compared to the dense layers.

VI. SUGGESTIONS

From the results acquired, we present some suggestions when applying autoencoder for feature extraction.

Autoencoder can be used to reduce the complexity of data, hence reducing the elapsed time and complexity of the machine learning model while increasing or maintaining its performance. It can be used in a pre−training step for preprocessing data before building machine learning models.

When using neural networks for machine learning tasks, there may be no need to use autoencoder for extracting features. Because deep model has a high capacity, which allows it to better understand features and data. As a result, feature extraction is somehow already included in the deep model, and it is likely to perform tasks well without data preprocessing. Hence, the effort here should focus on designing a powerful architecture to solve these problems.

The increase in k allows more information to be encoded in the latent code. Therefore, the machine learning model using extracted data is likely to perform better. As shown in Figures 2 and 5, there are some points of k at which the data is encoded in the latent space so well that it makes the model remarkably efficient. However, the encoded information becomes saturated gradually. At that point, increasing k has little effect on the performance of the model.

There are two reasons why the IVAE works well in our experiments. The first reason is the MMD object function, which is described by Zhao et al. [13]. The second reason is the architecture of the network. Specifically, it employs convolutional layers in the autoencoder, which show an advantage on the image data, as shown in Section 5.4.

VII. CONCLUSIONS

Autoencoders as well as other feature reduction methods are used in feature engineering to extract useful information from the raw data. They can minimize human effort and help to automate the data preprocessing process. Moreover, the information extracted by the autoencoder can then be used to improve or maintain the performance and reduce the complexity of machine learning models.

In this study, we reviewed feature extraction methods and autoencoders. Subsequently, we discussed different types of autoencoders and classifier. We then presented our methodology and conducted experiments using autoencoders with different values of k and classifiers on different datasets. Subsequently, we visualized and analyzed the results to assess the impact of these factors on the autoencoder and suggested the application of autoencoder for feature extraction.

For future research, it is worth investigating the behavior of the autoencoder to propose a novel autoencoder model. Additionally, we would like to develop an end-to-end approach for finding the best suited autoencoder for each specific task in which we no longer need to consider intermediate steps and only focus on the task.

REFERENCES

[1] Y. Bengio, A. Courville and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence,* vol. 35, no. 8, p. 1798-1828, Aug., 2013

[2] B. Ghojogh, M. N. Samad, S. A. Mashhadi, T. Kapoor, W. Ali, et al., "Feature selection and feature extraction in pattern analysis: A literature review," *arXiv preprint arXiv:1905.02845,* May, 2019

[3] M. Tschannen, O. Bachem and M. Lucic, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069,* Dec., 2018

[4] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016

[5] P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096-1103, Helsinki, Finland, Jul., 2008

[6] K. Sohn, X. Yan and H. Lee, "Learning Structured Output Representation Using Deep Conditional Generative Models," *Proceedings of the 28th International Conference on Neural Information Processing Systems*, vol. 2, pp. 3483-3491, Dec., 2015

[7] J. Walker, C. Doersch, A. Gupta and M. Hebert, "An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders," *eprint arXiv:1606.07873,* Jun., 2016.

[8] T.-V. Dang, H.-T. Vo, G.-H. Yu, J.-H. Lee, H.-T. Nguyen and J.-Y. Kim, "Removing Out-Of-Distribution Samples on Classification Task," *Smart Media Journal,* vol. 9, no. 3, pp. 80-89,

Sep., 2020

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114,* Dec., 2013

[10] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908,* Jun., 2016

[11] H. Kim and A. Mnih, "Disentangling by Factorising," *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 2649-2658, Stockholmsmässan, Sweden, Jul., 2018

[12] R. T. Q. Chen, X. Li, R. B. Grosse and D. K. Duvenaud, "Isolating Sources of Disentanglement in Variational Autoencoders," *NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems,* pp. 2615-2625, Dec., 2018

[13] S. Zhao, J. Song and S. Ermon, "Infovae: Information maximizing variational autoencoders," *arXiv preprint arXiv:1706.02262,* Jun., 2017

[14] A. Gretton, K. Borgwardt, M. J. Rasch, B. Scholkopf and A. J. Smola, "A Kernel Method for the Two-Sample-Problem," *Advances in Neural Information Processing Systems,* vol. 19, pp. 513-520, Dec., 2006.

[15] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644,* Nov., 2015.

[16] B. Gayathri and C. Sumathi, "An automated technique using Gaussian Naive Bayes classifier to classify breast cancer," *International Journal of Computer Applications,* vol. 148, no. 6, pp. 16-21, 2016

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning,* vol. 20, no. 3, pp. 273-297, Sep., 1995.

[18] G. James, D. Witten, T. Hastie and R. Tibshirani, *An introduction to statistical learning,* Springer, 2013

[19] L. Breiman, "Random forests," *Machine learning,* vol. 45, no. 1, pp. 5-32, Oct.., 2001

[20] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, Nov., 1998

[21] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations (ICLR)*, 2015

[22] Y. LeCun, C. Cortes and C. Burges, "MNIST handwritten digit database," *ATT Labs,* 2010

[23] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *CoRR,* 2017

[24] A. Krizhevsky, "Learning multiple layers of features from tiny images," Apr., 2009.

[25] Y. Bengio, E. Thibodeau-Laufer, G. Alain and J. Yosinski, "Deep Generative Stochastic Networks Trainable by Backprop," *ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning*, vol. 32, pp. 226-34, Jun., 2014

[26] T. Salimans, D. P. Kingma and M. Welling, "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap," *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol. 37, pp. 1218-1226, Jul., 2015

[27] T. D. Kulkarni, W. F. Whitney, P. Kohli and J. Tenenbaum, "Deep Convolutional Inverse Graphics Network," *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, vol. 2, pp. 2539-2547,

Dec., 2015

[28] D. J. Rezende, S. Mohamed and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models," *Proceedings of the 31st International Conference on Machine Learning*, vol. 32, no. 2, pp. 1278-1286, Jun., 2014

[29] K. Gregor, I. Danihelka, A. Graves, D. Rezende and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," *Proceedings of Machine Learning Research*, vol. 37, pp. 1462-1471, 2015.

[30] D. P. Kingma, D. J. Rezende, S. Mohamed and M. Welling, "Semi-Supervised Learning with Deep Generative Models," *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems,* vol. 2, pp. 3581-3589, Dec., 2014

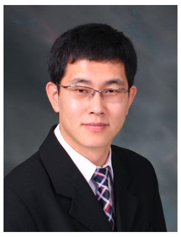
[31] S. Pant, J. Kim and S. Lee, "A Fall Detection Technique using Features from Multiple Sliding Windows," *Smart Media Journal,* vol. 7, no. 4, pp. 79-89, Dep., 2018

[32] T. D. Vu, H.-J. Yang, L. N. Do and T. N. Thieu, "Classifying Instantaneous Cognitive States from fMRI using Discriminant based Feature Selection and Adaboost," *Smart Media Journal,* vol. 5, no. 1, pp. 30-37, Jan., 2016

Authors

Kien Mai Ngoc

He received the B.S. degree in information technology from University of Engineering and Technology, Vietnam National University, in 2019. He is a Master student in Department of Data and HPC Science at University of Science and Technology (UST) and in Research Data and Sharing Center at Korea Institute of Science and Technology Information (KISTI).



Myunggwon Hwang

He received the B.S. degree in computer engineering, the M.S. degree in computer science, and the Ph.D. degree in computer engineering from Chosun University. He is a senior researcher in Center of Intelligent Infrastructure Technology Research at Korea Institute of Science and Technology Information (KISTI) and a professor in Department of Data and HPC Science at University of Science and Technology (UST). His research focuses on machine learning, text mining, training data selection, and knowledge acquisition.