The University of Sydney

School of Computer Science

Assignment 2 Report

Author:
520638064
520026168
510423153

Supervisor:
Irena Koprinska

An Assignment submitted for the UoS:

COMP5318 Machine Learning and Data Mining

May 19, 2023

Abstract

The objective of this project is to classify the EMNIST-ByClass dataset by using three different machine learning methods and to identify a best performed algorithm by comparing the evaluation metrics. The classification methods used in this report are K-Nearest Neighbors (KNN), MultiLlayer Perceptron (MLP) and Convolutional Neural Network (CNN) algorithm. The performance of these three algorithms are evaluated using different metrics including accuracy, precision, recall and confusion matrix. The ResNet-18 is considered as the most appropriate algorithm fitting in solving selected datasets classification problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The objective of this report is to build three classification models based on EMNIST dataset [Cohen et al., 2017] and choose a best performed algorithm by analyzing the evaluation results. The models constructed are K-Nearest Neighbors (KNN), Multi-Llayer Perceptron (MLP) and Convolutional Neural Network (CNN), which will be specifically explained in the methodology section. The performance of these models will be evaluated using accuracy, precision, recall and confusion matrix, and the comparison of evaluation results among the models will be discussed in the experimental section.

1.2 Background

Classification algorithms aim to correctly classify new observations into several categories, according to learning from a given dataset. The dataset selected in this report is the EMNIST dataset, an extension of MNIST, which is derived from NIST Special Database 19. This dataset can be seen as a standard benchmark for learning classification systems, it constitute a challenging classification tasks containing a set of handwritten character digits which has converted in 28x28 pixel image format.

1.3 Problem statement

This report will use the variant of EMNIST-ByClass dataset, where the dataset is split into an *emnist_train* set and an *emnist_test* set. The *emnist_train* set will be used in model training and the *emnist_test* set will be used in model evaluation. The whole dataset will be conducted by pre-processing, initial analysis, feature engineering. The processed *emnist_train* set will be used to construct the models, and the models will be adjust to the ideal state by tuning the hyperparameters. The obtained models will be evaluated by using the processed *emnist_test* set, and the best model will be selected by comparing the evaluation results.

Chapter 2

Methodology

2.1 Data Pre-processing

In the data pre-processing phase, several techniques were applied to prepare the EMNIST-ByClass dataset for classification. Firstly, the dataset was subjected to a flattening process, converting each image into a one-dimensional array, to ensure compatibility with the K-Nearest Neighbors (KNN) classifier. Additionally, Principal Component Analysis (PCA) was performed on the dataset to reduce the dimensionality of the input features for the KNN model. This step not only facilitated faster computation but also retained the essential information required for classification [Li et al., 2018].

Moreover, the dataset was normalized before training the Multi-Layer Perceptron (MLP) model. Normalization is a crucial data pre-processing technique that scales the features to a standard range, ensuring that all input variables contribute equally during training [Sharma et al., 2018]. This step enhances the convergence of the MLP model and prevents dominance by any particular feature.

For the Convolutional Neural Network (CNN) algorithm, the dataset was maintained in its original image format. As the ResNet-18 model was employed, the images were not flattened but rather treated as multidimensional inputs. The ResNet-18 architecture is specifically designed to handle image data and has demonstrated superior performance in image classification tasks [He et al., 2016].

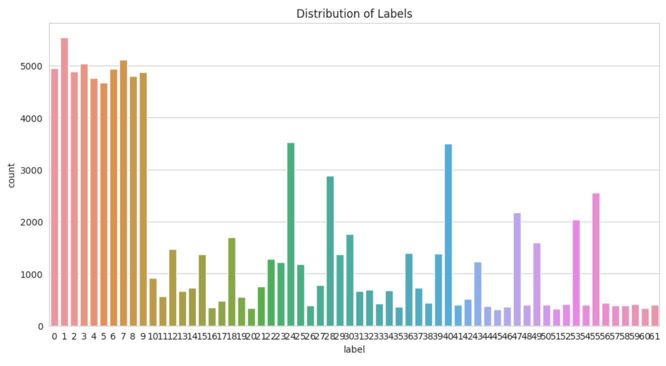


Figure 2.1: Imbalanced Data

As depicted in the illustrated Figure 2.1, the dataset suffers from the issue of data imbalance. To address this issue, different techniques were applied to each of the classification methods.

2.2 Methods

Three machine learning methods were utilized for classifying the EMNIST-ByClass dataset: K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN).

2.2.1 K-Nearest Neighbors (KNN)

The KNN algorithm is a simple yet effective classification method. It assigns a test sample to the majority class of its k nearest neighbors based on a distance metric [Altman, 1992]. In this study, KNN was applied on the flattened dataset after performing PCA, which reduced the dimensionality of the feature space [Belur et al., 2020]. To handle the issue of class imbalance, class weighting was used in the KNN model. The classweight parameter in scikit-learn's KNeighborsClassifier was set to 'balanced', assigning higher weights to the minority class samples, thus giving them more influence during the classification process.

2.2.2 Multi-Layer Perceptron (MLP)

The MLP model, implemented using the Keras library [Chollet, 2015], is a feedforward neural network with multiple hidden layers. For this project, the MLP consisted of two hidden layers. To tackle the class imbalance problem, undersampling was applied to the dataset before training the MLP model. The RandomUnderSampler class from the imbalanced-learn library [Lemaitre et al., 2017] was utilized to randomly remove samples from the majority class, balancing the class distribution.

2.2.3 Convolutional Neural Network (CNN)

The ResNet-18 model, a variant of the CNN architecture [He et al., 2016], was selected for its effectiveness in image classification. Unlike KNN and MLP, the ResNet-18 model operates on the original image data, leveraging its ability to extract meaningful features hierarchically through convolutional layers and residual connections. In addition to data augmentation techniques like random horizontal flips, rotations, and resizing, oversampling was employed to address the class imbalance issue. Synthetic samples from the minority class were generated using image augmentation techniques, effectively increasing the representation of the minority class in the training set.

These three methods were chosen based on their theoretical foundations and their suitability for the task at hand. KNN is a simple and interpretable algorithm that can handle high-dimensional data, while MLP and CNN are capable of learning complex nonlinear relationships in the data. Each method possesses unique characteristics that make it well-suited for the classification of the EMNIST-ByClass dataset.

2.3 Implementation

The implementation process involved several aspects, including code organization, libraries utilized, modularity, functions, classes, and data structures.

For the KNN algorithm, the scikit-learn library was utilized for its comprehensive set of machine learning functionalities. The code was structured to include the necessary import statements, initialization of the KNN classifier, and the implementation of the PCA technique for dimensionality reduction. Additionally, the code employed appropriate data structures to store the dataset and the trained KNN model. Modularity was ensured by encapsulating the code into functions, allowing for easy reusability and maintainability.

The MLP model was implemented using the Keras library, which provides a high-level interface for building and training neural networks. The code organization involved importing the required modules, defining the architecture of the MLP model, compiling it with the chosen optimizer and loss function, and fitting the model to the normalized dataset. To handle the class imbalance, undersampling was performed using the RandomUnderSampler class from the imbalanced-learn library. The Keras library's modular nature allowed for the creation of classes and functions to enhance code readability and reusability.

For the implementation of the ResNet-18 model, a deep learning framework like PyTorch was employed due to its support for building and training complex neural networks. The code structure followed a similar pattern, with necessary imports, model definition, optimization parameters, and the training process. The ResNet-18 model was instantiated, and the dataset, in its original image form, was used for training. Data augmentation techniques, such as random horizontal flips, rotations, and resizing, were applied to increase the diversity of the training samples. Additionally, oversampling techniques were employed to balance the class distribution by generating synthetic samples from the minority class.

In summary, the implementation process encompassed organizing the code, utilizing appropriate libraries, ensuring modularity through functions and classes, and leveraging suitable data structures. These practices facilitated the efficient execution and maintainability of the chosen methods for classifying the EMNIST-ByClass dataset.

Chapter 3

Experiments and Discussion

3.1 Experimental Setting

3.1.1 Dataset

The EMNIST-ByClass dataset provided by the assignment is the subset of the original EMNIST-ByClass dataset, which is one of the split of EMNIST datasets that includes all the 62 classes. The provided dataset contain two data files, which are train data and test data. Both of them are stored in the pickle format. The train dataset has 100000 data, while the test dataset has 20000 data. Each of them contains two keys, $data$ and $labels$. The $data$ is the a $numpy$ array with the shape $(dataset\ scale,\ 1,\ 28,\ 28)$, representing the numbers of grayscale images of size $28x28$ pixels. The $labels$ is a numpy array with the shape $(datasetscale,\ )$, containing the corresponding class labels for the test images. They are integers in the range of $[0,\ 61]$, representing the 62 classes in the EMNIST by_class dataset. The 62 classes include 26 uppercase letters $(A-Z)$, 26 lowercase letters $(a-z)$, and 10 digits $(0-9)$.

3.1.2 Model Architecture

The first K-nearest neighbors model is simply achieved by the $KNeighborsClassifier()$ in $Scikit-learn$. The parameters of *n_neighbors* is set to 1 initially. To find the best hyperparameters of this model, the *n_neighbors* list of $[1,\ 3,\ 5]$ and $weights$ list of $['uniform',\ 'distance']$ are used with the $GridSearchCV()$ in $Scikit-learn$ to find the best hyperparameter.

The second model Multilayer Perceptron (MLP) simply use $Keras$ to append one flatten layer $keras.layers.Flatten()$ and three fully connected layers $keras.layers.Dense()$ to the stack $keras.models.Sequential()$. The flatten layer flattens the image in shape of $(28,\ 28)$ to the $(784,\ )$ in order to pass the fully connected layer. The first connected layer has 500 *n_hidden_neurons* with the $tanh$ activation function, while the second connected layer has 300 *n_hidden_neurons* with the $tanh$ activation function. The third connected layer is the output layer with 62 *n_hidden_neurons* indicating the 62 classes of the data and $softmax$ activation function, which converts the output into a probability distribution. For hyperparameter tuning, this model is packaged in a function with the parameters of $n\_hidden\_neurons$ and $activation\_function$. This model is pass the $GridSearchCV()$ through the $KerasClassifier()$ to find the best hyperparameter in the lists [100, 300, 500] and ["*relu*", "*sigmoid*", "*tanh*"].

The third model is the ResNet-18 which is one of the CNN architecture. It accepts input images of size (28, 28) pixels with a single channel. Then, the initial convolutional layer with 64 filters, a kernel size of (7, 7), and a stride of (2, 2) is followed by batch normalization, *relu* activation function and a max pooling layer performing max pooling with a pool size of (3, 3) and a stride of (2, 2). After this, eight residual blocks, each consisting of two convolutional layers and batch normalization, utilize skip connections to learn residual functions. The input tensor is added element-wise to the output of each residual block from 64 to 512. Finally, a global average pooling layer reduces the spatial dimensions of the feature maps to a single value per feature map and a fully connected layer with 62 units (assuming it corresponds to 62 classes) and a $softmax$ activation function is applied for classification. To find the best hyperparameters, the $batch\_size$ in list of [32, 64, 128] and $activation$ in list of ["*relu*", "*sigmoid*", "*tanh*"] are chosen as the grid search. Because the other parameters like the filters, kernel and the residual block are specific value of the ResNet-18. The method of the hyperparameter tuning are same as the MLP.

3.1.3 Hardware and Software Specifications

The performance of models are evaluated on a laptop with the hardware and software following,

- $Laptop$: MacBook Pro (14inch 2021)
- CPU: Apple M1 Pro
- *CPU Core*: 10-Core
- *GPU Core*: 16-Core
- RAM: 16GB
- $Compiler$: Jupyter Notebook

The packages with versions are shown below,

- $Python$ 3.9.13
- $numpy$ 1.23.5
- $matplotlib$ 3.7.1
- $scikit-learn$ 1.2.2
- $pickle$ 4.0.0
- $pandas$ 1.5.3
- $seaborn$ 0.12.2

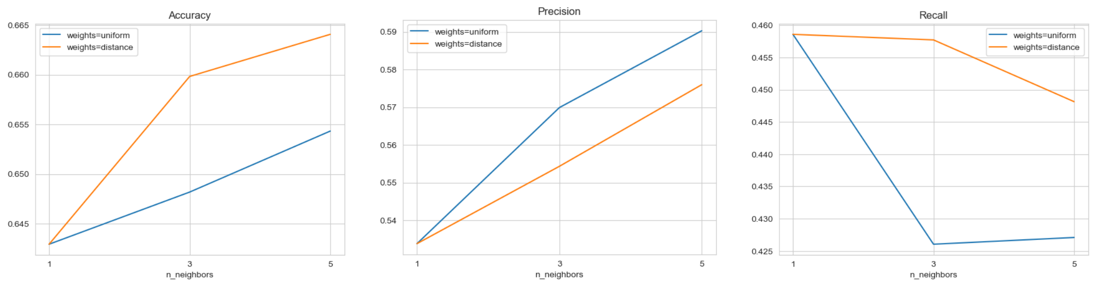
- $tensorflow$ 2.12.0
- $keras$ 2.12.0
- $scikeras$ 0.10.0

3.2 Experimental Results

3.2.1 KNN

The KNN is the easiest and the fast model to complete the classification task. The dataset for this model is the dataset after *PCA* because the KNN classifier can only accept the flatten image data. The *PCA* can also decrease the runtime.

To find the best hyperparameters of the KNN classifier, the model compares different values of *n_neighbors* and *weights* with the value of $[1,\ 3,\ 5]$ and $['uniform',\ 'distance']$. The accuracy, precision and recall are selected as the compare metrics with the results below,



(a) Accuracy (b) Precision (c) Recall

Figure 3.1: KNN Hyperparameter Comparison

The detailed results of the Figure 3.1 are shown in the following table,

Accuracy	**1**	**3**	**5**	Precision	**1**	**3**	**5**	Recall	**1**	**3**	**5**
uniform	0.642	0.647	0.654	**uniform**	0.534	0.570	0.591	**uniform**	0.458	0.426	0.427
distance	0.642	0.660	0.664	**distance**	0.534	0.555	0.575	**distance**	0.458	0.457	0.448

Table 3.1: KNN Hyperparameter Comparison

According to the Figure 3.1 and Table 3.1, it is easy to find that the best hyperparameters of the KNN are the 5 *n_neighbors* and *distance weights*. Therefore, the best performance of the KNN model are trained with these hyperparameters and the final results are shown below,

Best Performance	**Accuracy**	**Precision**	**Recall**
5 n_neighbors distance	69.580%	60.505%	48.875%

Table 3.2: Best KNN Hyperparameter Results

From the Tabel 3.2, we can see that all metrics are not very high because the KNN is a very simple classifier. The confusion matrix of the best KNN are visualized in the Figure 3.2,

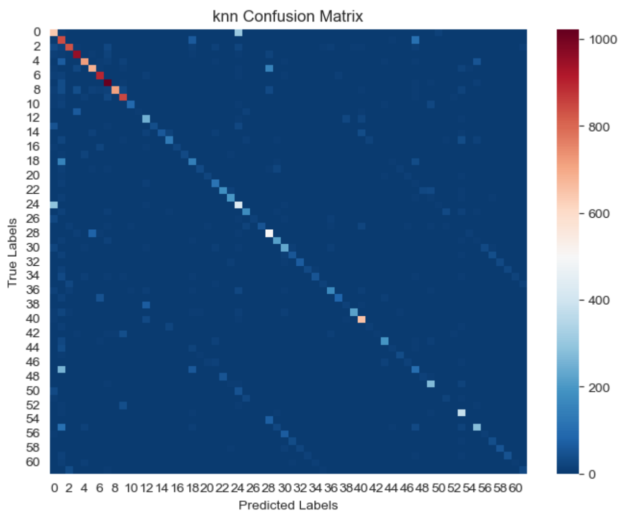
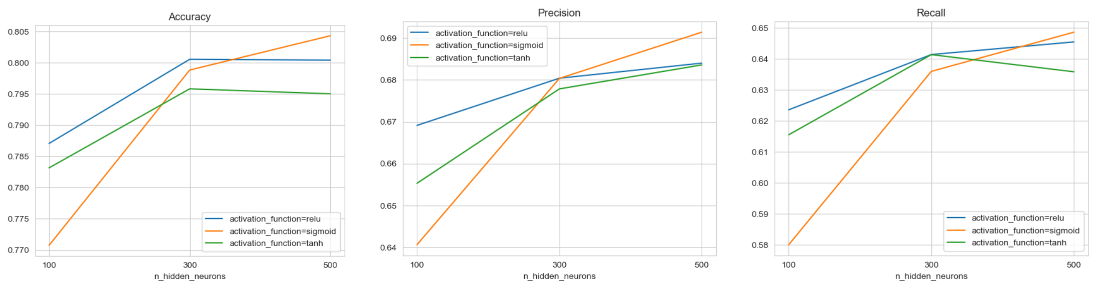


Figure 3.2: Best KNN Confusion Matrix

3.2.2 MLP

The MLP model is achieved by the *Keras* with 2 hidden layers, whichi is a simple neurual network. The dataset with the normalization are trained with this model.

To find the best hyperparameters of the MLP, the model compares different values of *n_hidden_neurons* and *activation_function* with the value of [100, 300, 500] and [”*relu*”, ”*sigmoid*”, ”*tanh*”]. All the training process are trained with 64 batch and 10 epochs in a same optimizer and loss function. The accuracy, precision and recall are selected as the compare metrics with the results below,



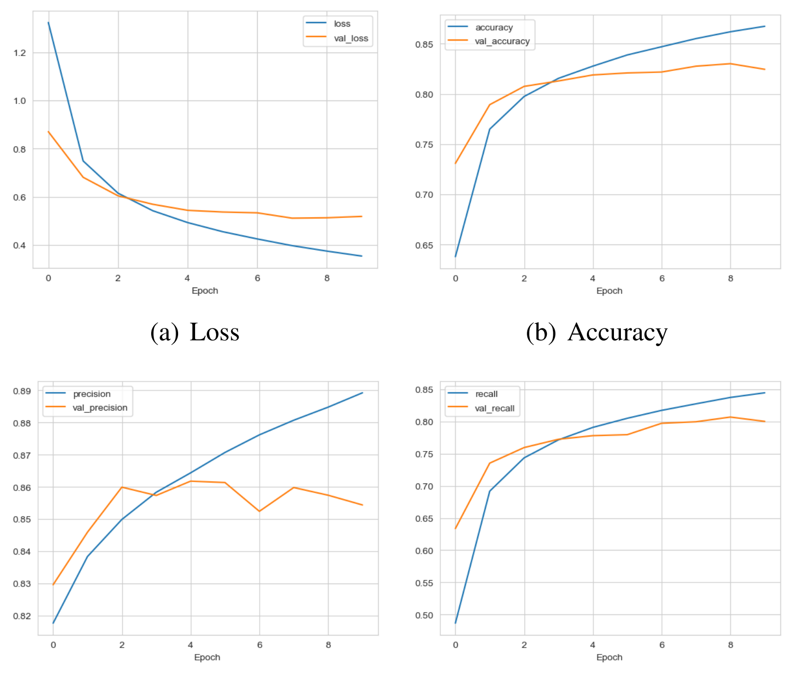
(a) Accuracy (b) Precision (c) Recall

Figure 3.3: MLP Hyperparameter Comparison

The detailed results of the Figure 3.3 are shown in the Table 3.3. According to the Figure 3.3 and Table 3.3, it is easy to find that the best hyperparameters of the MLP are the 500 *n_hidden_neurons* and *sigmoid activation_function*. Therefore, the best performance of the MLP model are trained with these hyperparameters and the final results are shown below,

Accuracy	**100**	**300**	**500**	Precision	**100**	**300**	**500**	Recall	**100**	**300**	**500**
relu	0.787	0.801	0.801	**relu**	0.669	0.680	0.684	**relu**	0.623	0.641	0.645
sigmoid	0.771	0.798	0.804	**sigmoid**	0.642	0.680	0.692	**sigmoid**	0.580	0.636	0.649
tanh	0.783	0.796	0.795	**tanh**	0.655	0.678	0.683	**tanh**	0.616	0.641	0.636

Table 3.3: MLP Hyperparameter Comparison



(a) Loss (b) Accuracy

(c) Precision (d) Recall

Figure 3.4: Best MLP Hyperparameter Performance

Best Performance	**Loss**	**Accuracy**	**Precision**	**Recall**
500 n_hidden_neurons sigmoid	0.5870	80.6650%	83.1636%	78.3900%

Table 3.4: Best MLP Hyperparameter Results

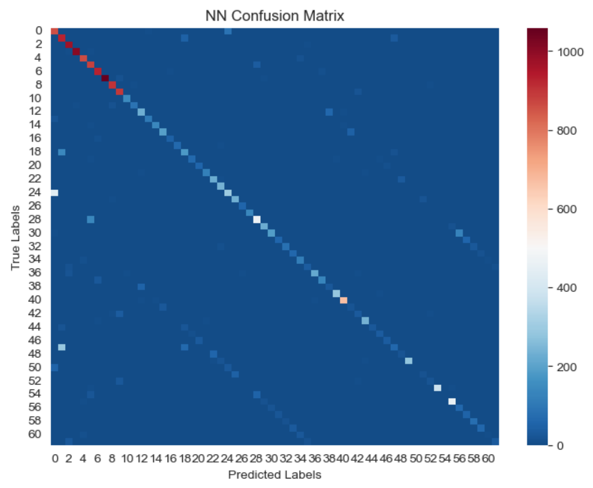


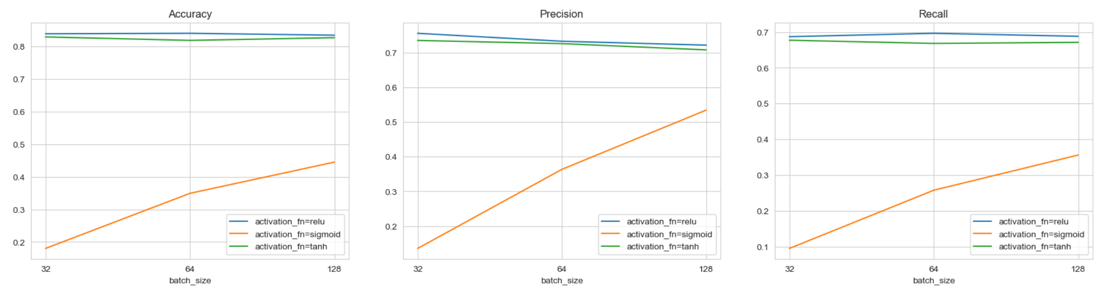
Figure 3.5: Best MLP Confusion Matrix

From the Figure 3.4 and Table 3.4, we can see that there is an overfitting of this model. Because the model is too simple too train the dataset. It does not perform well on the validation data. The confusion matrix of the best MLP are visualized in the Figure 3.5.

3.2.3 ResNet-18

The ResNet-18 model is one of the efficient architecture of the CNN. Therefore the trained dataset is in the shape of image. It is a complex model for the classification task.

To find the best hyperparameters of the ResNet-18, the model compares different values of $batch\_size$ and $activation\_fn$ with the value of [32, 64, 128] and [”*relu*”, ”*sigmoid*”, ”*tanh*”]. All the training process are trained with 10 epochs in a same optimizer and loss function. The accuracy, precision and recall are selected as the compare metrics with the results below,



(a) Accuracy (b) Precision (c) Recall

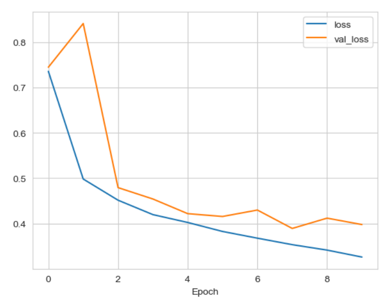
Figure 3.6: ResNet-18 Hyperparameter Comparison

The detailed results of the Figure 3.6 are shown in the following table,

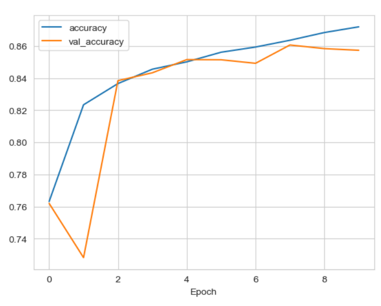
Accuracy	**32**	**64**	**128**	Precision	**32**	**64**	**128**	Recall	**32**	**64**	**128**
relu	0.83	0.84	0.83	**relu**	0.75	0.73	0.72	**relu**	0.69	0.70	0.69
sigmoid	0.16	0.35	0.45	**sigmoid**	0.14	0.36	0.53	**sigmoid**	0.10	0.25	0.35
tanh	0.82	0.83	0.82	**tanh**	0.74	0.72	0.71	**tanh**	0.68	0,67	0,68

Table 3.5: ResNet-18 Hyperparameter Comparison

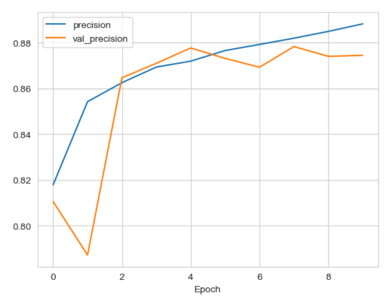
According to the Figure 3.6 and Table 3.5, it is easy to find that the best hyperparameters of the ResNet-18 are the 64 $batch\_size$ and $relu$ $activation\_fn$. Therefore, the best performance of the ResNet-18 model are trained with these hyperparameters and the final results are shown below,



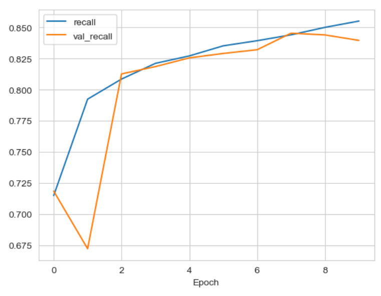
(a) Loss



(b) Accuracy



(c) Precision



(d) Recall

Figure 3.7: Best ResNet-18 Hyperparameter Performance

Best Performance	**Loss**	**Accuracy**	**Precision**	**Recall**
64 batch_size relu	0.4238	85.0800%	87.0469%	83.0950%

Table 3.6: Best ResNet-18 Hyperparameter Results

From the Figure 3.7 and Table 3.6, we can see that the performance of ResNet-18 is better than others. Because the limitation of the hardware, the epochs can not be long. But It does not have an obvious overfitting. The confusion matrix of the best ResNet-18 are visualized in the Figure 3.8,

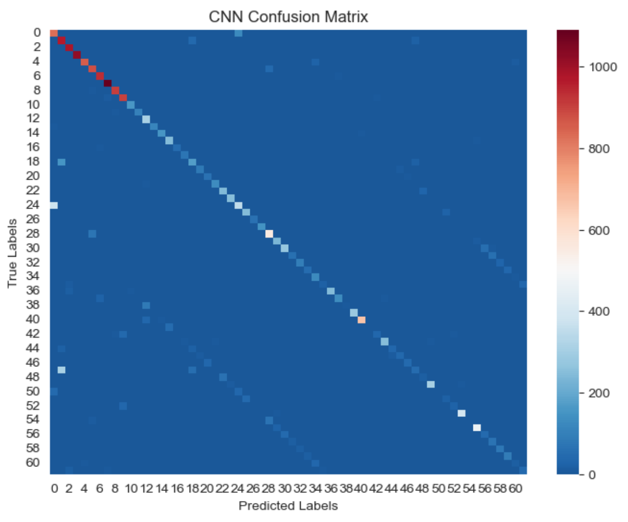
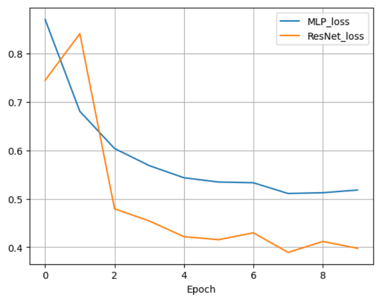


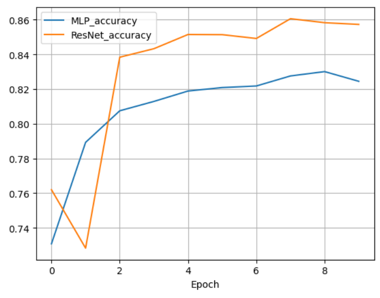
Figure 3.8: Best ResNet-18 Confusion Matrix

3.3 Comparison

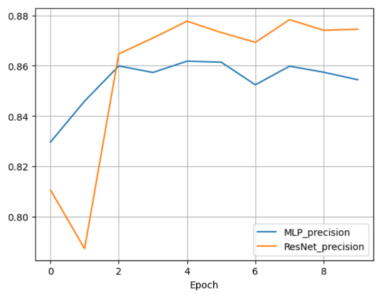
All three models have the best hyperparameters. However the performance of these models are different. Because the $KNeighborsClassifier()$ does not have the concept of the epoch. Therefore, the performance of the validation data for MLP and ResNet-18 are compared in figures firstly. In addition, it is obvious that the MLP and ResNet-18 perform better than the KNN from the above sections. The performance of training process are shown below,



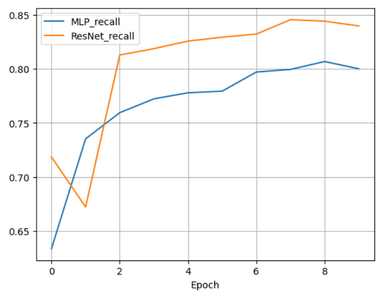
(a) Loss



(b) Accuracy



(c) Precision



(d) Recall

Figure 3.9: Different Models Comparison

Obviously, all performance of the ResNet-18 perform better than the MLP about 5 percents according to the Figure 3.9. This situation is expected because the ResNet-18 is a better architecture to handle the image dataset. In addition, the complexity of it is the highest one above these three models.

After training the models, the test dataset is tested by these three models. The Comparison of the loss, accuracy, precision and recall on test set are shown below. The cost time on training process in 10 epochs are also included in it.

	Loss	**Accuracy**	**Precision**	**Recall**	**Training Time**
KNN	0.9174	69.580%	60.505%	48.875%	3.45min
MLP	0.5870	80.665%	83.163%	78.390%	0.67min
ResNet-18	0.4238	85.080%	87.047%	83.095%	48.07min

Table 3.7: Different Models Results

According to the Table 3.7, all performance of the ResNet-18 are better than others models. However, the ResNet-18 also costs the longest time. The reason is that, the CPU are used for the training. Therefore, it will cost many time. If using the GPU as the training device, the ResNet-18 will save more time and the performance will be better. The MLP are only about 5 percents smaller than the ResNet-18, while it saves much time. The KNN performs the woreset in this task, because all metrics are less than 70 percents.

3.4 Reflection

First of all, to improve all performance of the three models, the *PCA* reduces the number of features and removes less informative components to improve computational efficiency. In addition, normalization helps algorithms converge faster and ensures fair treatment of features.

KNN is a simple yet effective algorithm for classification tasks. It works well when there are local patterns in the data and can capture similarities between samples. Since the dataset EMNIST-ByClass is a classification task, KNN can provide a good baseline model for comparison with more complex methods like MLP and ResNet-18. By constract, MLP is selected to improve performance by leveraging its ability to learn complex non-linear relationships between input features and target outputs. As EMNIST-ByClass is a complex task with multiple classes, MLP can learn higher-level representations and extract more abstract features from the data, enabling it to better discriminate between different classes.

To improve the performance, the *Adam* or *RMSprop* are used to help accelerate convergence and improve training speed on the one hand. On the other hand, experimenting with different hyperparameter settings, such as the number of hidden layers, hidden units, learning rate, and activation functions, are used to help find the optimal configuration for the MLP.

Therefore, we proposed the ResNet-18 model to further enhance performance due to its ability to handle deep architectures and alleviate the vanishing gradient problem. ResNet-18 has shown excellent performance in various image classification tasks, and EMNIST-ByClass, which deals with handwritten characters, can benefit from its feature extraction capabilities. Processing multiple samples in parallel using batch processing can improve efficiency and exploit parallelization capabilities.

In summary, the chosen methods are motivated by their respective strengths in capturing local patterns, learning complex relationships, and handling deep architectures. We make efforts to improve performance involving techniques such as hyperparameter tuning, regularization, and optimization algorithms.

Chapter 4

Conclusion and Future Work

4.1 Summary

In conclusion, this report has explained the methodology of three classification algorithms which are KNN, MLP and ResNet-18. These three algorithms were applied in this report to construct models based on the variant of EMNIST-ByClass dataset including 62 classes of handwritten digits.

The dataset was firstly analyzed and investigated by exploratory data analysis (EDA), and its main characteristics was summarized by employing data visualization methods. In this process, the labels of target value were changed to the one-hot mode, and the pixel values of input were normalized to be between 0 and 1. The principle components analysis was applied to reduce the dimensionality of the data, and the data was scaled to have zero mean and unit variance. This process is to obtain clean data to obtain better prediction results in trained models.

Further, three models were constructed using the processed data, and the performance of each algorithm has been evaluated by using different metrics including accuracy, precision, recall and confusion matrix. The first model is KNN algorithm which can be considered as a benchmark to compare with other complex models, as it is a simple algorithm which can effectively solving classification models. The best state of KNN is when its hyperparameters are tuned to 5 neighbors and using distance weights, its accuracy, precision and recall are around 69.6%, 60.5%, 48.9% respectively. The second model is MLP, which can learn high-level representations and extract abstract features in multiclassification tasks. It performed best when tuning its hyperparameters to 500 hidden neurons and using *sigmoid* activation function, its accuracy, precision and recall are around 80.7%, 83.2%, 78.4% respectively. The third model is ResNest-18, which has advantage in constructing deep networks as it overcomes the vanishing gradient problem. The best performance of ResNet-18 is when tuning its hyperparameters to 64 batch and using *relu* activation function, its accuracy, precision and recall are around 85.1%, 87%, 83.1% respectively.

Overall, the ResNet-18 can be considered as the best performed algorithm in conducting the classification task based on selected dataset, as its evaluation result is obviously better than other models. Specifically, the accuracy, precision and recall of ResNet-18 are approximately 5% higher than MLP and much higher than KNN. This might be because that ResNet-18 is appropriate in handling the image dataset and benefits from its feature extraction advantage. However, compared with the other two models, an obvious drawback of ResNet-18 is that the model is more complex and takes longer to train. This might be because that the equipment used in training model is central processing unit (CPU), which is better at performing small-scaled neural networks. CPUs are less efficient at processing massive data, which might lead to increases training time as the number of layers and parameters increase.

4.2 Future Work

In considering the limitations of ResNet-18, its high computational complexity and high memory requirements should be taken into account. In dealing with this, an equipment with higher computing power can be used in future study. This can contributes to further scale the hyper-parameters of models and reach the better state of models. Specifically, the utilization of graphics processing unit (GPU) in training ResNet-18 algorithm might be more appropriate. This is because that GPU can provide tremendous acceleration for neural network, it provides the parallel processing which can support to simultaneously compute tasks with greater speed and efficiency. As the neural networks are designed to run in parallel and each task running independently, GPU might be better suited for processing the massive datasets used in neural networks training.

Bibliography

[Altman, 1992] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

[Belur et al., 2020] Belur, A., Yujing, W., and Aljohani, M. (2020). *K-nearest neighbor (KNN) classification*, pages 277–288. Springer.

[Chollet, 2015] Chollet, F. (2015). Keras. GitHub repository.

[Cohen et al., 2017] Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Lemaitre et al., 2017] Lemaitre, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

[Li et al., 2018] Li, X., Bao, Y., Xiao, J., Hu, B., and Gao, S. (2018). Emnist: An extended mnist dataset for handwritten character recognition. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE.

[Sharma et al., 2018] Sharma, A., Sharma, P., and Garg, S. (2018). A comparative analysis of classification algorithms. *International Journal of Advanced Research in Computer Science*, 9(1):382–386.

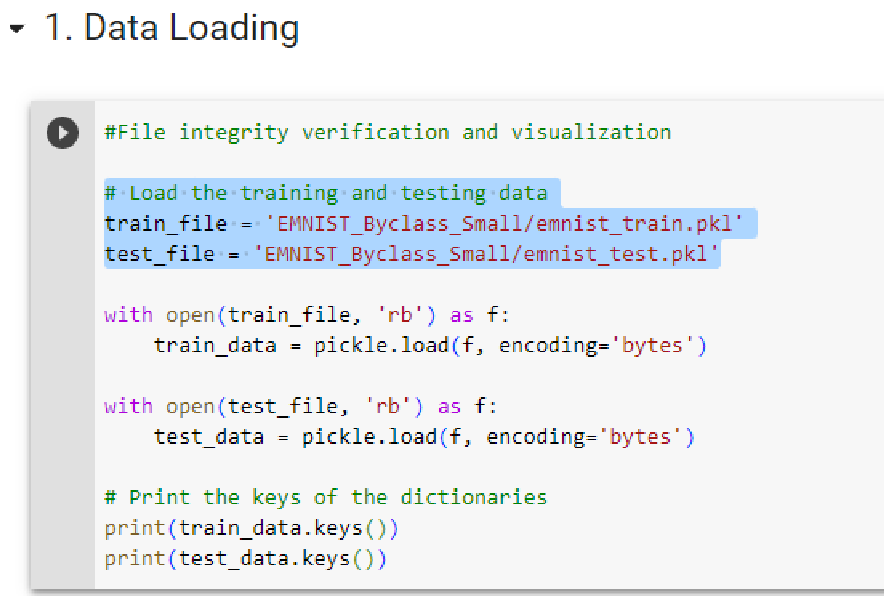
Appendix A

Instruction for Code

The code is in the format of the *ipynb*.

Assuming the Jupyter Notebook has been installed on the computer, you just need to run block by block in the Jupyter platform.

Then, you should change the data file path in the *Data Loading* cell. The file path should be the data path in your computer as showing below. The data files are the link from the assignment 2 specification.



1. Data Loading

```
#File integrity verification and visualization

# Load the training and testing data
train_file = 'EMNIST_Byclass_Small/emnist_train.pkl'
test_file = 'EMNIST_Byclass_Small/emnist_test.pkl'

with open(train_file, 'rb') as f:
    train_data = pickle.load(f, encoding='bytes')

with open(test_file, 'rb') as f:
    test_data = pickle.load(f, encoding='bytes')

# Print the keys of the dictionaries
print(train_data.keys())
print(test_data.keys())
```

Figure A.1: File Path

There are several required packages to run the code. If you don't have the specific package, you should run the **!pip install package_name** in the cell. Two uncommonly used packages have been installed in the first cell using **! pip install scikeras** and **! pip install seaborn**. The required packages and libraries are shown below,

- *Python* 3.9.13
- *numpy* 1.23.5

- $matplotlib$ 3.7.1
- $scikit-learn$ 1.2.2
- $pickle$ 4.0.0
- $pandas$ 1.5.3
- $seaborn$ 0.12.2
- $tensorflow$ 2.12.0
- $keras$ 2.12.0
- $scikeras$ 0.10.0