Assignment 2 - Group: 11

Abstract

This assignment utilizes the EMNIST handwritten character dataset to accomplish the classification task. We firstly learned about the basic information of the original dataset through selecting some images of the dataset randomly, drawing a pairplot graph and scatter plot to analyze the relationship between these features. According to these basic information of the dataset, we chose the appropriate techniques like data augmentation and normalization to preprocess the data. Then we used CNN to predict the data and customized other two different architectures such as ResNet-like architecture and EfficientNet-like architecture based on the common ResNet-18 architecture and EfficientNet-B0 architecture to make the model adapt to the small and intermediate dataset of this assignment. After hypertuning the parameters of three different models, we compared the performance of those models based on accuracy, f1_score, precision and other classification metrics to get the best model of them by visualizing the result. Finally, we found that ResNet-like architecture has the best performance with around 86% accuracy of this classification task.

1. Introduction

The dataset chosen in this assignment is EMNIST, which is a set of handwritten characters derived from NIST Special Database 19. As it is the extension of MNIST dataset, this dataset can be converted to a 28x28 pixel image format [1]. Unlike the MNIST dataset, EMNIST is more challenging for classification tasks with different categories of letters and numbers. It is compatible with the current classifiers and its structure of data can match with that of MNIST dataset [2]. This classification task of the EMNIST dataset is actually a handwritten characters recognition problem because the model we use in this task must identify these characters precisely to predict the class labels correctly. Therefore, to improve the performance of the model in this classification task, it is critical to improve the ability of models to identify the handwritten characters.

Whatever in the past or in recent years, the research of character recognition problems has a practical significance and application value to the current society. For example, the Optical Character Recognition (OCR) is widely used in many fields such as digitization of printed articles, robotics and traffic surveillance [3]. These applications indicate that the technology of OCR has become an important tool to promote the process of digitization. Apart from that, character recognition technology is applied to deal with and index the text information in images and videos in the multimedia database [4]. Therefore, the research of character recognition problems plays an important role in our society. This assignment of handwritten characters classification tasks can provide a good referenced method for related character recognition subjects.

In the previous years, for the classification task of related character recognition problems such as classification of MNIST or EMNIST usually use the simple convolutional neural network (CNN) or a more complex network built on the top of the original CNN by superimposing convolution and pooling layers. For example, Singh et al. [5] proposed an artificial neural network (ANN) which was composed of "three convolutional neural networks (CNNs), two fully connected neural networks (FCNs), three max-pooling blocks (MPs), four rectified linear units (ReLUs) and one flatten layer". This architecture was a more complex network based on the simple CNN and got an excellent performance of classification tasks. Additionally, the research of Brown [6] introduced data augmentation and transfer learning to improve the generalization ability of the model and avoid the problem of overfitting. The baseline model still used the CNN architecture but added additional multiple normalization layers to improve the training efficiency and stability of the model.

According to the previous research, we find that simple CNN can already give a relatively good prediction. So in this assignment, we still use the simple CNN as a baseline model to accomplish this classification task and use it to compare with other deep learning methods. Moreover, to further improve the result of prediction, we consider choosing those special architectures like ResNet-18 and EfficientNet-B0. However, considering that the dataset we use in this assignment is small and intermediate, we have made lightweight improvements to these special architectures and customize two architectures called ResNet-like and EfficientNet-like respectively to make them more applicable to this task. Meanwhile, we also combine the preprocessing techniques such as data augmentation and normalization that were commonly used in the previous research to make the model more robust.

2. Data

2.1 Data description and exploration

The dataset we use in this assignment is the small version of the EMNIST dataset, which is an extension of the MNIST dataset and has a set of handwritten characters. It is derived from NIST Special Database 19 and its data can be converted to 28x28 pixel images format [1]. The dataset includes a train set which has 100000 examples and a test set which has 20000 examples. The original shape of the train set is (100000, 1, 28, 28), representing 100,000 grayscale images of size 28x28 pixels, while the original shape of the test set is (20000, 1, 28, 28), representing 20,000 grayscale images of size 28x28 pixels. There are totally two keys which are ‘data’ and ‘labels’ in the dataset. After flattening the three-dimensional image data into two-dimensional data, whatever the train set or the test set, both of them have 62 classes and 784 features. In order to more conveniently evaluate the different deep learning methods, we extract 10% of the train set as the validation set for the process of evaluation.

Before preprocessing the data, we firstly choose the data exploration to know some basic information of the dataset. Firstly, we select 10 images of the original data randomly to have a preliminary understanding of the image. From the image shown below (Figure 1), the characteristic of the dataset is that the majority of characters have distinct differences with others. However, there is still a difficulty that some characters are not clear to identify, the “label 6” has some similarity with the “label 40” , which may make the model misclassify the labels and affect the precision of the model. Moreover, some characters such as “label 1” may not have the proper orientation, which also brings a challenge for the model to classify correctly.

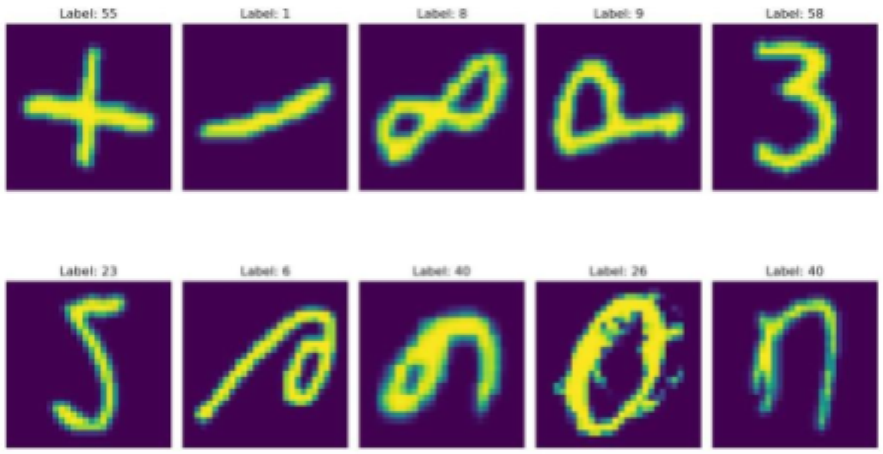

Figure 1. images randomly selected from original dataset

Then we draw the pairplot graph of the first three features to further explore the data and observe the distribution and correlation between features. According to the image shown below (Figure 2), we find that the distribution between features is highly concentrated, and there is no obvious linear relationship between them. Due to the weak linear relationship between features, using the linear methods such as linear regression or support vector machine (SVM) may not work well in the classification task for this dataset.

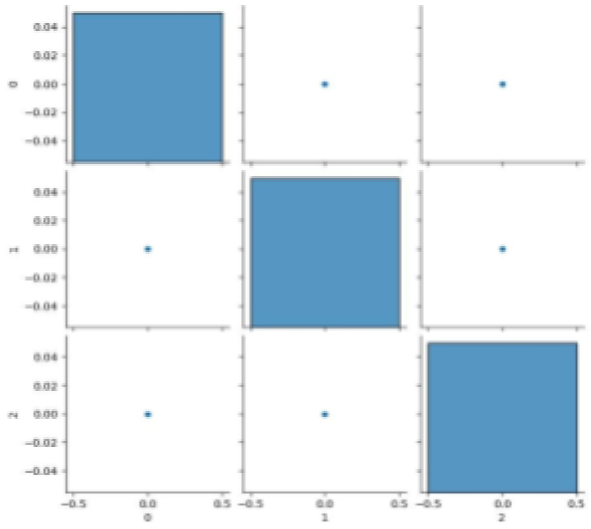

Figure 2. Pairplot graph

Moreover, we also draw the scatter plot of the first two features to understand the distribution of features more intuitively. As shown in the graph (Figure 3), the data point is deep-concentrated in the central position of the graph, which indicates that the first two features are poorly differentiated. Combined with the pairplot graph mentioned above, we find that the first few features have no obvious differentiation. Based on this finding, we suppose that there may also be little correlation between features of the whole dataset and traditional linear methods may not have a good performance in this task, it is more appropriate to choose non-linear methods like CNN to accomplish this task. Moreover, according to the basic information reflected by the first few features, we may need the normalization techniques for data preprocessing to enhance the degree of differentiation between features and improve the quality of the input data of the model.

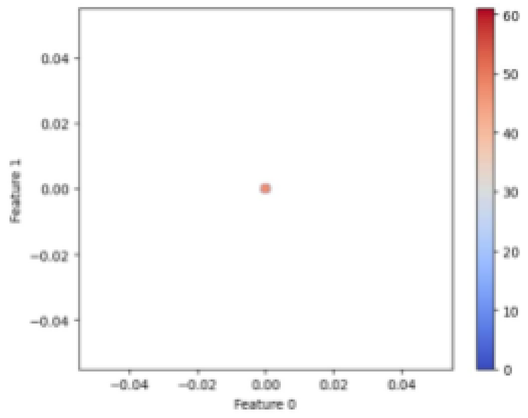

Figure 3. Scatter plot

2.2 Pre-processing

As mentioned above, there are some characters of the original dataset that have a problem that they don't have the proper orientation, which may lead to the precision degradation of the model. So we choose the data augmentation to preprocess the data through rotating and flipping the images. For the dataset like EMNIST, using ways of rotating and flipping to augmentate the data can make the model better learn the representation of these characters in different directions, thus improving the abilities of the model to identify the characters with various orientations. Additionally, Brown [6] reported that data augmentation can effectively help models prevent overfitting even though the model may use the deeper network and improve the robustness of the model. Therefore, data augmentation is appropriate in the data preprocessing phase.

In the previous data exploration, we found that there may not be obvious differentiation between some features. Actually, when the range of values for some features are much larger than that of other features, the model may tend to assign higher weights to the features with larger range of values and ignore those with lower range of values. This problem can be worse under the circumstance of features with no obvious differentiation. So we choose normalization technology to ensure balanced training. Normalization usually transforms the numerical data into the special range, typically between (0, 1) to make the model prevent the problem of bias in training due to the features having different ranges of values [7]. The formula of the normalization is as follows:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \quad (1)$$

For this dataset we use in the assignment, normalization is important because the the pixel values of these handwritten characters are usually from 0 to 255, normalizing the pixel values into the range between 0 and 1 not only can accelerate the convergence rate but also help the model better learn the small difference between those features to improve the abilities of identifying the characters correctly.

After normalizing the data, we once decided to use dimension reduction techniques to preprocess the data. As mentioned above, this dataset may not be appropriate to use some linear methods to process the data. So originally, we chose a non-linear method called Uniform Manifold Approximation and Projection (UMAP) to reduce the dimensions. UMAP is a technique for dimension reduction "based on a fuzzy topological analysis of data" [8]. UMAP usually arranges the data in the low-dimension space by graph layout algorithms. The algorithm searches the projection of the data that has the closest possible equivalent global shape and structure of the original dataset in the low-dimension space to find the embedding and preserve the topology of data [9]. In this process, UMAP can preserve the global structure of the data and realize the purpose of dimension reduction. Actually, when we chose UMAP to reduce the dimensions at first, we didn't get the desired result. As shown below (Figure 4), we found that after dimension reduction, the data points of different categories are not classified explicitly, which may lead to decrease the performance of the model for this task. Actually, when we preserved the result of UMAP for

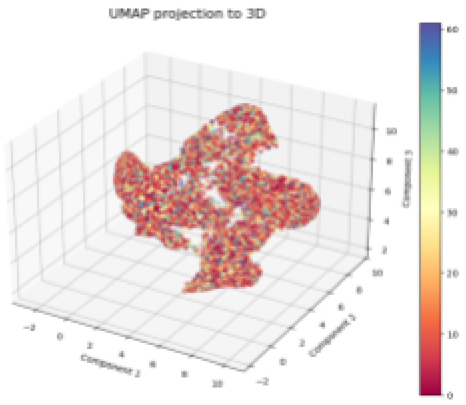

Figure 4. UMAP projection to 3D

the next classification task, the accuracy of the model was very low. Meanwhile, we also used other dimension reduction methods like PCA, t-SNE to reduce the dimensions and used these data to train the model. We also got unsatisfactory results. Using the data after the above dimension reduction methods to train the model all resulted in low classification precision of the model. So we gave up using UMAP in the preprocessing phase eventually. Overall, the final preprocessing technologies that we use are data augmentation and normalization.

3. Methodology

3.1 Model1:ResNet-like

Theory: To ease the training of deeper networks, the residual learning framework was introduced. The core idea behind the ResNet architecture is the introduction of residual blocks, where instead of learning the desired underlying mapping directly, the network learns the residual, which is the difference between the input and output [10]. Residual Learning enables the network to learn simpler transformations by focusing on residuals F(x) = H(x) - x , instead of directly learning the full transformation H(x)**.**

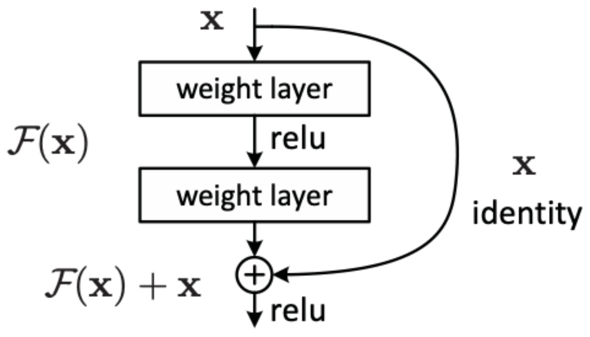

Figure5. residual block [10]

Reason: We chose ResNet because its introduction of residual blocks simplifies the training process and enables networks to achieve greater depth. Also, it can improve accuracy significantly in many image classification tasks.

Strengths: The skip connections ensure that faster convergence can be achieved, as the gradients can flow more directly through the model and vanishing gradients are bypassed, which also leads to faster training and therefore lower costs for the use of resources; Due to the special structure, ResNet can learn more general structures in the data and would not focus on dataset-specific features, which increases the generalization of the model and delivers better results with unseen data [11].

Weaknesses: The skip connections lead to a higher complexity of the model, which can be reflected in higher computing requirements; With small data sets, ResNet can also lead to overfitting, because the model structure is too complex and cannot be sufficiently learned with the few training data.

Architecture and hyperparameters:

The architecture of the ResNet-like model we introduced closely follows the ResNet structure with some adjustments and simplification to suit the task. The architecture begins with a convolutional layer that uses a kernel size of 7x7 and a stride of 2 to capture more detailed features from the 28x28 input images.

Batch normalization and ReLU are followed. After these, a max pooling layer is used to reduce the dimensions and prevent overfitting. Multiple residual blocks are then stacked with increasing channel sizes (from 64, 128, 256 to 512) to increasingly capture more complex features. Each residual block is followed by a dropout layer to help prevent overfitting. Then, we introduce global average pooling to further downsample the features. The final dense layer reduces the outputs to match the 62 classes with a softmax activation function.

In this model, we focus on tuning three key hyperparameters to optimize the performance: learning rate, dropout rate, and the number of epochs. To identify the best combination of these hyperparameters, we choose to use Grid Search with cross-validation. We search over learning rates of 0.1, 0.01 and 0.001 to control how quickly the model updates its weights. A higher rate speeds up convergence but risks instability, while a lower rate makes more stable but slower learning. We tested dropout rates of 0.2, 0.3, and 0.4 to prevent overfitting. Higher rates provide stronger regularization, but lower rates may use more learning capacity. We varied epochs between 10 and 20 to control how long the model trains. More epochs give the model more learning time but also increase the risk of overfitting.

3.2 Model: simple CNN

Theory: The main idea of a simple Convolutional Neural Network is to extract features using local filters and weight sharing for efficient image recognition. CNNs are composed of five types of layers: the input layer, convolutional layers, pooling layers, fully connected layers, and the output layer.

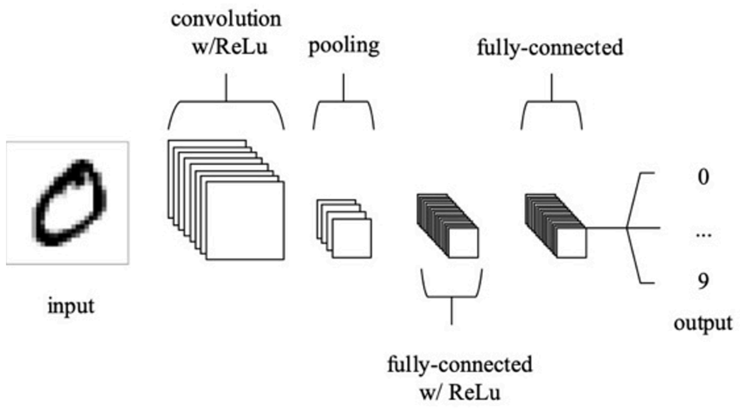

Figure 6. basic structure of simple CNN [12]

Reason: We choose the simple CNN model because it primarily focuses on solving difficult images recognition tasks and it is well-suited for the EMMIST dataset. Also, it helps us evaluate the more complex models' performance by comparing to which of this simpler architecture.

Strengths: Simple CNN excels at image-related tasks because of its ability to capture spatial hierarchies in the data [13]; In simple CNN, the same filter is applied across different parts of the image, which significantly reduces the number of parameters, making it more efficient than traditional ANNs.

Weaknesses: Simple CNN's limited capacity can lead to issues like underfitting (too simple to capture the underlying patterns in the data), overfitting on small datasets (learns too much from the training data, including noise and irrelevant details), and limited feature extraction capabilities.

Architecture and hyperparameters:

The Simple CNN architecture we developed consists of two convolutional layers, each followed by max pooling and dropout layers, and concludes with fully connected layers. The architecture starts with two convolutional layers: the first one applies 32 filters of size 3×1 to the input image (28×28×1) with a ReLU activation; and the second one applies 64 filters of size 3×1 with ReLU. After each convolutional layer, we add a MaxPooling layer with a 2×1 pool size one by one. After each pooling layer and the fully connected layers(with 128 neurons and ReLU), we incorporated a Dropout layer. Following the second convolutional block (convolution + pooling + dropout), the 2D output is flattened into a one-dimensional vector. This architecture can balance simplicity and great performance by using two convolutional layers to extract essential features and max-pooling layers to reduce dimension and prevent overfitting. Dropout is used to further regularization, while the fully connected layers and softmax output ensure effective multi-class classification.

In this model, the key hyperparameters we focus on and the method we use is similar to what we have done in the model1. We choose learning rate, dropout rate, and the number of epochs and use Grid Search with cross-validation. The difference is that, we search over learning rates of 0.01 and 0.001; test dropout rates of 0.1, 0.2, and 0.3; varied epochs between 20 and 25.

3.3 Model3: EfficientNet-like

Theory: EfficientNet is a model developed to address the limitations of traditional CNNs by proposing a more structured scaling approach. The core theoretical foundation of the EfficientNet is its new scaling method, which uniformly scales all dimensions of depth, width and resolution using a simple yet highly effective compound coefficient [14]. This ensures that the network grows in a balanced way and optimizes performance without unnecessary computational costs.

$$\begin{aligned} \textbf{depth: } & d = \alpha^{\phi} \\ \textbf{width: } & w = \beta^{\phi} \\ \textbf{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Figure7. compound Scaling Formula for EfficientNet [14]

Reason: The core scaling method of EfficientNet lets this model maximize accuracy while minimizing computational demands, which is well-suited for limited computational power.

Strengths: By using compound scaling, it manages to achieve a high level of accuracy with fewer FLOPS compared to other deep learning models; The EfficientNet model has a big family, from EfficientNet-B0 to EfficientNet-B7, making it easy for people to select suitable architecture that fits different requirements.

Weaknesses: While EfficientNet has strong performance, its architecture is less interpretable than others, because of the complexity of the compound scaling process.

Architecture and hyperparameters:

The architecture of the EfficientNet -like model we introduced closely follows the EfficientNet structure with some adjustments and simplification to suit the task. The architecture begins with a stem block, which contains a convolutional layer of 32 filters with a kernel size of 3 and strides of 2, followed by batch normalization and swish activation. After that, we employ a series of Mobile Inverted Bottleneck Convolutional Blocks (MBConv blocks), starting with 16 filters and progressively increasing to 320 filters. In the final layers, the Conv2D layer with 1280 filters and a kernel size of 1 is applied, followed by batch normalization and swish activation. The final dense layer reduces the outputs to match the 62 classes with a softmax activation function.

In this model, we focus on tuning three key hyperparameters to optimize the performance: learning rate, survival probability, and batch size. To identify the best combination of these hyperparameters, we choose to use Grid Search with cross-validation. We search over learning rates of 0.01, 0.001 and 0.0001 to control how quickly the model updates its weights. A higher rate speeds up convergence but risks instability, while a lower rate makes more stable but slower learning. We tested survival probability of 0.8, 0.7, and 0.6 to adjust the stochastic depth feature. Higher survival probabilities make the network more consistent in structure but can lead to less regularization while lower survival probabilities adding regularization and potentially improving generalization. We varied batch size between 128 and 256 to balance the trade-off between faster training and memory consumption.

4. Experimental Results

4.1 Experimental Setup

Dataset: EMNIST dataset is a set of handwritten characters. Its data can be converted to 28x28 pixel images format and includes a train set which has 100000 examples and a test set which has 20000 examples.

Models: as described in part 3.

Hardware, and software specifications of the computer: as described in the appendix.

4.2 Result evaluation and comparison

We compare the performance of the selected three models: ResNet-like, Simple CNN, and EfficientNet-like, using key evaluation metrics like accuracy, precision, recall, F1-score, and AUC-ROC. We use the high quality plot (Figure 8) and table (Table 1) below to more intuitively demonstrate the related classification metrics of these three deep learning methods and observe the different hyperparameters that are used in these models.

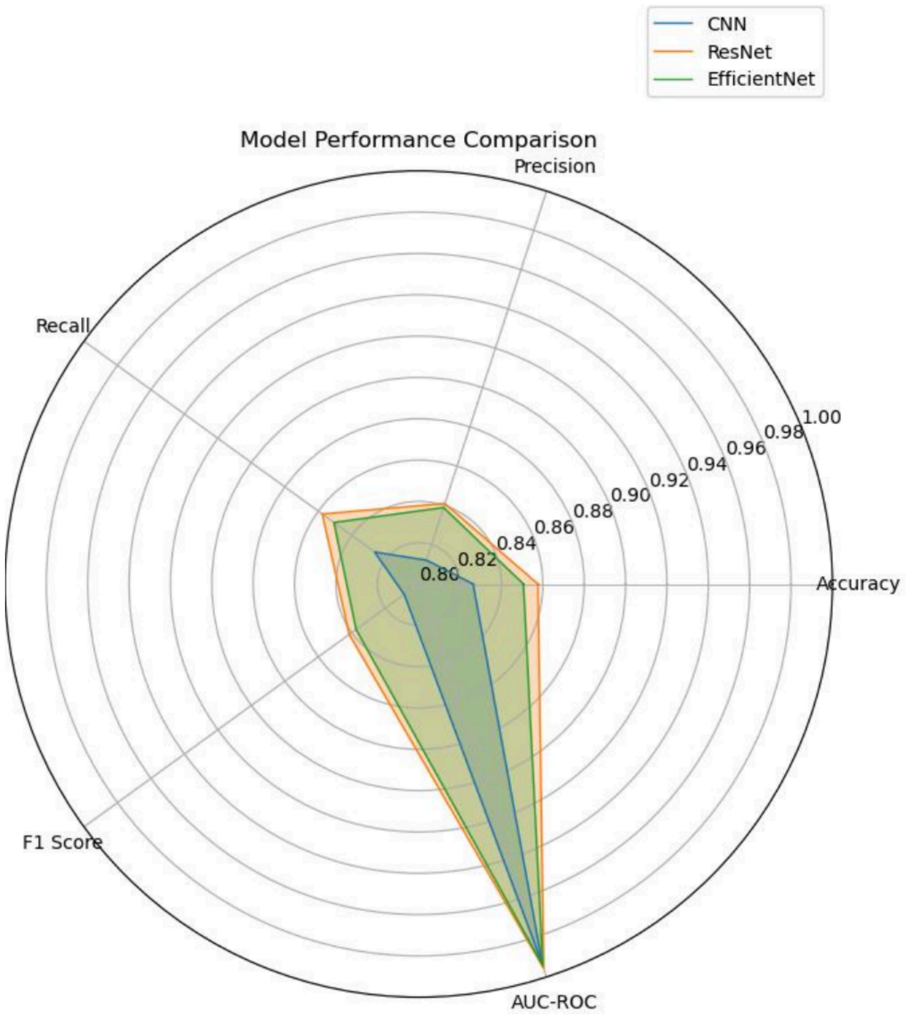

Figure8. the radar chart for model performance comparison

Each model was fine-tuned using different hyperparameters.

Table1. hyperparameters of three models

	ResNet-like	Simple CNN	EfficientNet-like
Learning rate	0.001	0.001	0.001
Survival-prob	/	/	0.8
Batch size	/	/	128
dropout	0.3	0.3	/
epochs	20	20	/
Best accuracy	0.8605	0.8208	0.8413
The number of hyperparameters	3	3	3

Based on the results and illustration, the ResNet-like model achieved the highest accuracy of 86.05%, while EfficientNet-like and Simple CNN followed closely with 84.13% and 82.08%, respectively. According to the radar chart above, it is clear to see that the ResNet-like architecture is better than the other two architectures in all the classification metrics which indicates that this model has better overall performance in dealing with such this kind of classification tasks and capture the features of data more precisely to identify and classify the classes. Radar chart provides a global view to demonstrate the advantages or disadvantages of each model in all the classification metrics.

According to the table above, the EfficientNet-like model used additional hyperparameters, survival-prob and batch size. The survival-prob parameters are introduced in the EfficentNet-like architecture, which stands for the survival probability. We use the survival probability in the MBConv blocks to help the EfficientNet-like model improve the robustness and abilities of generalization. The varying hyperparameters configurations in these models demonstrate their different abilities to process the dataset and achieve optimal performance.

In summary, the ResNet-like model is the best classifier in this task. The introduction of the residual blocks makes this model identify the characters more precisely and the lightweight design makes this model adapt to the small and intermediate dataset, thus not only reducing the training time of the model but also preserving excellent accuracy of the prediction.

5. Conclusion

After comparing all the deep learning methods and visualizing the result, we found that ResNet-like architecture had the best performance with the accuracy around 86%, 0.86 precision, recall and F1_score. However, according to the classification report, all the models can't identify some labels correctly, typically for the "label 50" and "label 54", which indicates that all the models still have a large limitation. These two labels are both characters that are difficult to identify, it is unexpected that the precision, recall, F1_score of three models for predicting these two labels are 0. We suppose that the imbalance of the original dataset leads to this result because if these characters are not very common in the original dataset, all the models can't learn enough features of these labels to predict them, thus affecting the precision of the model.

In fact, according to the overall result of prediction, these models successfully accomplish the classification tasks. All the models can predict the majority of labels correctly and acquire a good performance. The preprocessing techniques such as data augmentation and normalization greatly improve the generalization abilities of the models. Moreover, compared with simple CNN, ResNet-like architecture and EfficientNet-like architecture display better performances and precisions for predicting the labels. Meanwhile, according to the limitation of these models, we still have some possible directions for further improvement of the models in the future. One direction is to introduce the oversampling techniques to deal with the imbalance of the dataset. Additionally, we will also consider more complex architectures like ResNet-32 and longer training time to make sure the model gets enough training time to learn those features and improve their abilities of identifying characters.

In this assignment, the most important thing we have learned is that selecting appropriate models for the task is critical. Initially we chose lots of inappropriate models such as RNN, LSTM and got undesired results, which wasted much time. However, after searching lots of journals, we found that the methods based on CNN were more suitable for this kind of classification task and finally completed this assignment. In the process of searching journals, we also learned so many special architectures like ResNet and EfficientNet, which significantly widened our scopes of knowledge and got a deeper understanding of deep learning methods.

References

[1] G. Gohen, S. Afshar, J. Tapson, A. van Schaik. (2017). *EMNIST: an extension of MNIST to handwritten letters* [Online]. Available: https://arxiv.org/abs/1702.05373

[2] A. Baldominos, Y. Saez, P. Isasi, “A Survey of Handwritten Character Recognition with MNIST and EMNIST,” *Applied Sciences*, vol. 9, no. 15, pp. 3169, July 2019.

[3] M. B. Bora, D. Daimary, K. Amitab, D. Kandar, “Handwritten Character Recognition from Images using CNN-ECOC,” *Procedia Computer Science*, vol. 167, pp. 2403-2409, 2020.

[4] N. Arica and F. T. Yarman-Vural, “An overview of character recognition focused on off-line handwriting,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 31, no.2, pp.216-233, August 2002.

[5] S. Singh, A. Paul, M. Arun, “Parallelization of digit recognition system using deep convolutional neural network on CUDA,” In *2017 IEEE Third International Conference on Sensing, Signal Processing and Security (ICSSS)*, pp.379-383, May 2017.

[6] D. Brown and I. Lidzhade, “Handwriting Recognition using Deep Learning with Effective Data Augmentation Techniques,” presented at the *2021 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pp.1-9, August 2021.

[7] P. V. Singh. (2023, Sep. 23). *All About Min-max scaling* [Online]. Available: All About Min-max scaling. Min-max scaling, also known as… | by Pooja Vivek Singh | Medium

[8] F. Trozzi, X. Wang, P. Tao, “UMAP as a Dimensionality Reduction Tool for Molecular Dynamics Simulations of Biomacromolecules: A Comparison Study,” *J Phys Chem B*, vol. 125, no. 19, pp. 5022-5034, May 2021.

[9] M. Vermeulen, K. Smith, K. Eremin, G. Rayner, M. Walton, “Application of Uniform Manifold Approximation and Projection (UMAP) in spectral imaging of artworks,” *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, vol. 252, pp.119547, May 2021.

[10] K. He, X. Zhang, S. Ren, J. Sun. (2015). *Deep residual learning for image recognition* [Online]. Available: https://arxiv.org/abs/1512.03385

[11] Databasecamp. (2023, Mar. 18). *ResNet: Residual Neural Networks – easily explained* [Online]. Available: https://databasecamp.de/en/ml/resnet-en

[12] K. O'Shea. (2015). *An introduction to convolutional neural networks* [Online]. Available: https://arxiv.org/abs/1511.08458

[13] S. An, M. Lee, S. Park, H. Yang, J. So. (2020). *An ensemble of simple convolutional neural network models for mnist digit recognition* [Online]. Available: https://arxiv.org/abs/2008.10400

[14] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” In *International conference on machine learning*, PMLR, pp.6105-6114, 2019.

Appendix

Software requirements

In this appendix, we will provide a way to set up the environment to run the code. We mainly use the anaconda to create the virtual environment to run the code.

Firstly, we should click the anaconda prompt from the anaconda navigator, just as the Figure 9 shown below:

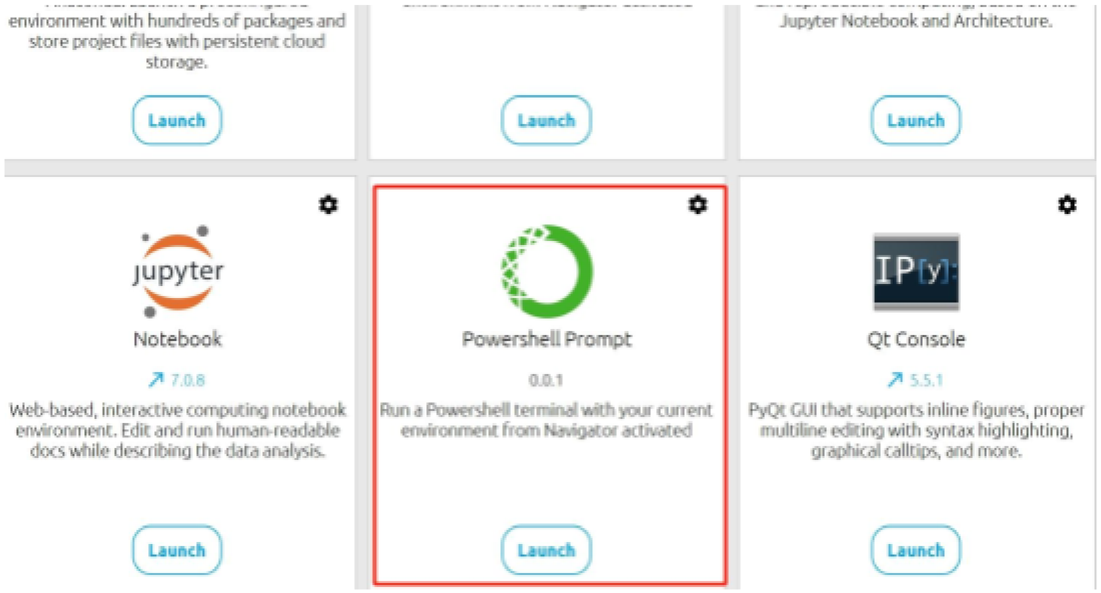
environment with hundreds of packages and store project files with persistent cloud storage.
Jupyter Notebook and Architecture.
Launch
Launch
Launch
jupyter
Notebook
↗ 7.0.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
IP[y]:
Qt Console
↗ 5.5.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Figure 9. launch the anaconda prompt

Then we should input the command in the anaconda prompt one by one:

```
conda create -n tf_gpu python=3.9
conda activate tf_gpu
pip install tensorflow-gpu==2.7
conda install -c conda-forge cudatoolkit=11.6
conda install cudnn=8.3
pip install protobuf==3.20
pip install numpy==1.23.4
pip install scikeras==0.6
conda install pandas
conda install matplotlib
pip install scikit-learn==1.0.2
conda install seaborn
```

After inputting the commands above, we successfully create the virtual environment called tf_gpu to run the code, the versions of those packages are as follows:

python = 3.9.19,   tensorflow-gpu = 2.7.0,   cudatoolkit = 11.6.2,   cudnn = 8.3.2.44, protobuf = 3.20.0,   numpy = 1.23.4,   scikeras = 0.6.0,   pandas = 2.2.2,   matplotlib = 3.9.2

scikit-learn = 1.0.2,   seaborn = 0.13.2.

Please make sure that all the versions of packages strictly obey the versions listed above to prevent the problem of version incompatibility between libraries.

Then we should install the ipykernel to make sure our jupyter notebook from anaconda can run the code in the virtual environment. Then we input the commands in the anaconda prompt as follows:

```
conda install ipykernel
```

```
python -m ipykernel install --user --name tf_gpu --display-name “Tensorflow”
```

Through these two commands, we add the ipykernel called “Tensorflow” to the Jupyter notebook from anaconda and now we can use this ipykernel to run our code.

We lauch the Jupyter notebook from anaconda navigator and choose the ipykernel “Tensorflow” in the Jupyter notebook, just as the Figure 10 shows:

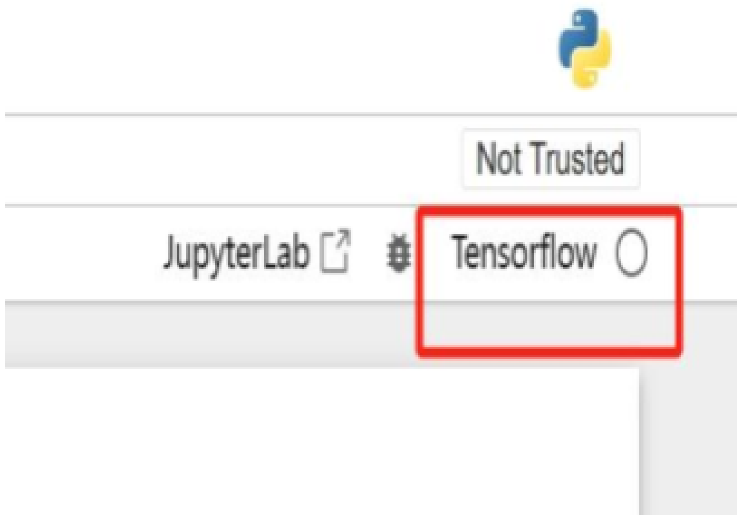

Figure 10. choose the kernel of the Jupyter notebook

Now we can successfully run the code in the notebook.

Hardware requirements

It is suggested that the code can be run in the following configuration:

CPU:

(1) Type: 13th Gen Intel Core i9-13900HX

(2) Number of Cores: 24

Memory: 32GB RAM

Storage: 1TB SSD

GPU:

(1) Type: NVIDIA GeForce RTX 4090

(2) Graphics Memory: 16G

Display: 17.3-inch, 2.5K resolution, 240Hz refresh rate

If the code can’t be run normally due to unsatisfying the hardware requirements of the computer, we can also use the cloud server such as Colab to run the code.