Assignment 2 - Group: <A2-Group 74>

Group members:

1. **540155277 Zifan_Hu**
2. **530783442 Liehao Song**
3. **490051481 Lihang Shen**

Abstract

This report examines the effectiveness of deep learning techniques on handwriting character recognition using the EMNIST dataset. Specifically, we implement and compare the performance of three models: a 2-layer Multi-Layer Perceptron (MLP), a 2-layer Convolutional Neural Network (CNN), and a 2-layer Long Short-Term Memory (LSTM) network. Among these, the LSTM model demonstrates superior performance. The findings highlight the robustness of deep learning models in accurately identifying handwritten characters.

Keywords

Deep learning, MLP, CNN, LSTM, EMNIST dataset, handwriting recognition.

1. Introduction

1.1 Introduce the problem and the dataset we chose

In this assignment, we choose the "EMNIST handwritten character dataset". The problem we choose to solve is about identifying those handwritten characters correctly. The reason we chose this data set is that we want to know why current technology can precisely detect human handwriting. Besides, we also believe this technology is matured which means we can find more related documents to help us to understand how to improve our own experiments.

1.2 discuss its relevance to real-world applications

Its applications are varied. For example, teachers can use it to grade students' homework. Softwares can automatically detect customers' handwritten words. Camera can automatically detect License plate. There are many others here that I won't go into.

1.3 Outline the previous technique and approaches that have been utilized to identify handwriting.

Before Deep learning was introduced, people always used machine learning methods. Such as SVM, which is one of the most powerful machine learning models for identifying pictures. Additionally, using KNN is another option, although it struggles to accurately identify shifted images. The performance for those methodologies are much much weaker than the deep learning due to some weakness. Like I mentioned before, most machine learning models cannot resolve shifted image problems which means they will classify shifted images as another class and that could lead to the accuracy loss and increase of training time.

1.4 Overview of the methods we used

In order to realize our goal, we need to choose our models first. Through discussion, we decide to use 2 layer MLP, CNN, and LSTM methods. The reason we choose these 3 is that we think those 3 models are covered in Tutorial materials; therefore, we can imitate TA's method to implement our own model. Secondly, we should train the model. Similar to assignment 1, we decided to split 20% of the train set as validation set and use the remaining 80% of training data as the training set. Besides, we also use the combination of cross validation and grid search to tune the hyperparameters. Since "grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid," (G. S. K. Ranjan, A. Kumar Verma and S. Radhika, 2019). In addition, all of us use the colab, therefore, we need not to worry about the efficiency too much, hence we choose to set the limit of possible combinations to less than 36. Then when we use a 3-cv method, we only need to train 108 models which is acceptable for us. After every group member finishes their model training, we just simply compare our results and choose the best model.

1.5 Key findings and results

In this project, we find that the precision and efficiency is usually conflicted with each other. If we want to get the best model with the best hyper parameters, we need to use much time for training and tuning, especially, it takes more than 9 hours for MLP methods to find the best combination of hyperparameters. In addition, we find all methods have similar performance. We think that is because all deep learning methods can easily handle image clarification. Finally, our result is that LSTM is the best model and it uses an "adam" optimizer, 0.2 drop rate, 150 neurons for the input layer, 75 neurons for the hidden layer, and 62 neurons for the output layer.

2. Data Description and Processing

2.1 Description and exploration of EMNIST

The EMNIST (Extended MNIST) dataset is an extended version of MNIST, developed and maintained by the National Institute of Standards and Technology (NIST) in the United States. (Cohen, 2017) This dataset provides a broad collection of handwritten characters, covering numbers (0-9) as well as uppercase and lowercase letters. It is mainly used for machine learning and pattern recognition tasks, especially in the fields of handwriting recognition and deep learning.

In this assignment, we use a small subset of EMNIST-ByClass dataset, which has 100,000 samples and two classes: 'data' and 'label'. The class 'data', which is the handwritten character digits, is converted to a 28x28 pixel image format. Handwritten characters in this dataset are shown in Appendix.

For this dataset, the most difficult part is that handwritten characters are often non-standard and varied. Compared to printed text, people's handwriting styles vary greatly, and the quality of handwritten text is poor, which poses a significant obstacle to converting it into machine-readable text. (Anli, 2022) Specifically, the challenges of handwriting recognition for our model are mainly including:

1. The style of handwritten strokes varies from person to person, and there is a huge ambiguity

2. Even the handwriting style of the same person will change from time to time and be inconsistent
3. The quality of the source document or image is usually very low, it may be blurry, dark, etc
4. The text in a printed document is on a straight line, but humans don't need to write a line of text on a white paper in a straight line
5. Cursive handwriting makes it difficult to separate and recognize characters
6. Handwritten text can be rotated to the right, which is in sharp contrast to all printed text with straight lines

We need to select appropriate models and design appropriate model structures based on these characteristics to overcome these difficulties and successfully extract key features of handwritten characters.

2.2 Data Preprocessing

Model1: LSTM

For the LSTM model, I choose Normalization for pixel value, dividing them by 255. The original data is from 0 to 255, after transformation, the data is from 0 to 1.

If the data is not scaled, we will meet many problems. Firstly, it's possible to meet exploding or vanishing gradients if the range of data is too large and numbers of data are too varied. Moreover, "it is possible for large inputs to slow down the learning and convergence of your network and in some cases prevent the network from effectively learning your problem." (Brownlee, 2019)

And data normalization can solve these problems, which will improve the accuracy of the model.

We also use "one-hot encoding" methods for the LSTM model. It's a method that converts numbers to binary vectors where only the corresponding position is 1 and others are 0. This method lets calculation of loss become easier, we can use categorical_crossentropy to calculate it. Moreover, it's easy to design and modify and allows a state machine to run at a faster clock rate than any other encoding of that state machine.(Wikipedia contributors, n.d.)

Model2: CNN

During the training process, apart from the test set, it is customary to split the training data into a training set and a validation set. This is to facilitate monitoring the model's performance on unknown data. Specifically, we used the train_test_split method to split the training set.

Considering that the convolutional layers of CNN models have strong feature extraction capabilities, especially when the input images are relatively consistent (such as handwritten characters), CNN can perform well without complex preprocessing. This is because CNN models do not require manual input of features, and their convolutional networks can automatically learn useful features from the raw data.

At the same time, the EMNIST dataset contains sufficient data volume and has a high diversity among character samples, which allows the model to learn representative and broad features from it, reducing the risk of overfitting. Even without additional preprocessing steps, larger datasets can help models perform well.

From the above analysis, it is not difficult to see that there is not much necessity for more complex preprocessing of the dataset. Moreover, in the actual training process, we found that without using appropriate data preprocessing methods, the preprocessed data can actually reduce the performance of the model. Therefore, we have decided not to use any further preprocessing methods for the training data of the CNN model, except for segmenting the training set and validation set.

Model3: MLA

As usual, I began by checking for any missing values in the dataset, and none were found. Following this, I proceeded with the normalization step. From the Week 7 tutorial materials, I learned that large input values can result in gradients close to zero, potentially causing the vanishing gradient problem. I opted not to use PCA for several reasons. First, MLPs (Multilayer Perceptrons) have the ability to automatically detect useful features. Second, while PCA can reduce dimensionality, it may also result in the loss of important information, even if the loss seems minimal. Finally, MLPs are capable of handling high-dimensional data without the need for dimensionality reduction techniques like PCA.

3. Methodology

3.1 Model1: LSTM

3.1.1 Theory

LSTM is one of RNN models. To describe LSTM, we should know what RNN is first. The RNN model includes three layers: input layer, hidden layer and output layer. The function of the RNN input layer is to read the input features for example word sequence of text. A hidden layer calculates h(t) by input x and h(t-1), and sends h(t) to the next position for calculation of h(t+1). Output layer takes h(t) as input and uses it to calculate the result of prediction. (Khan & Qamar, 2020)
And LSTM is specifically designed to address the limitations of traditional RNN, it can decide whether a date will be retained or forgotten, in order to solve the Vanishing and Exploding Gradient problem of traditional RNN.
I chose this model because it's a typical RNN architecture model, and I want to justify how well a RNN structure can be in a classification job. Moreover, LSTM is suitable for dealing with nonlinear data relationships, which means that it can have a relatively high performance in this task.

3.1.2 Strengths and weaknesses

The strengths of LSTM are high performance on sequential data, less prone to overfitting, good at handling nonlinear dynamics and robust gradient flow.
The reason is that LSTM has a special feature: gating mechanism (which I will introduce in next step). It can help LSTM control information flow, understand text over time better, and learn complex pattern better so that decrease the extent of overfitting, understanding complex, nonlinear data better and their cell state, which maintains a more stable gradient flow during backpropagation, can help this model deal with vanishing gradient problem.
However, the gating mechanism also has some weaknesses. Firstly, the model has relatively high computational cost, as a result, training LSTMs can be time-consuming. Moreover, it has more parameters and increases the risk of overfitting when the dataset is small. But our dataset is really large and we don’t need to worry about it.

Except for the weaknesses caused by the gating mechanism, the model also has another weak point: difficult to interpret. It's a "black box", we can't see how it works inside, it's harder to understand and explain the learned patterns in it.

3.1.3 Architecture and hyperparameters

Firstly, I will explain the gating mechanism. It concludes with three gates: the Input gate, which decides what new information should be added to the cell state. The next one is Forget gate, deciding which information in the cell state should be forgotten. The last one is Output gate, it decides the information to be passed to the next hidden state and output (Zhao, et.al., 2017)
With these layers, we can selectively remember and forget information, allowing them to capture long-range dependencies in the data, and let the model perform better on complex and nonlinear situations.
Now I would like to describe each layer I use in my model. The first layer is a LSTM layer. This layer has 28x28 size input and n units, the number of units in this layer is one of hyperparameters that need to be tuned.
The next layer is a dropout layer, it randomly drops out some input units in order to prevent overfitting, the rate of dropout is another hyperparameter.
Then we add the second LSTM layer, and the number of units is half of the first LSTM layer, processing these sequences further.
We also have two dense layers, while the first dense layer uses ReLU activation for non-linear transformations and has the same number of units with the second LSTM layer. The second one uses softmax activation tailored for multi-class classification across, and the dimension of output is 62, because it uses one-hot encoding to present the output and there are 62 kinds of labels in our dataset. The picture of my model is in appendix, figure 9.
The values for n_units to search are 50, 100, 150, for drop_out rate are 0.1, 0.2, 0.3, and for optimizer are 'adam', 'rmsprop'.

3.2 Model2: CNN

3.2.1 Theory

Convolutional Neural Network (CNN) is a deep learning model specifically engineered for processing image data and is extensively employed in image recognition, object detection, and other computer vision undertakings. The architectural design of CNN is inspired by the hierarchical structure of human visual processing, being highly suitable for capturing hierarchical patterns and spatial dependencies within images. (Zoumana, 2023) It encompasses multiple layers, such as convolutional layers, pooling layers, and fully connected layers.

The principal components of convolutional neural networks primarily consist of:

Convolutional layer: This constitutes the initial building block of CNN and is also its core layer. These layers implement convolution operations on the input image, utilizing filters (also referred to as kernels) to detect features like edges, textures, and more complex patterns. Convolution operations assist in maintaining the spatial relationships among pixels.

Activation function: Nonlinear activation functions, like the corrected linear units (ReLU), are prevalently utilized as activation functions to introduce nonlinearity into the model, enabling it to better represent intricate image features.

Pooling layer: The pooling layer downsamples the spatial dimensions of the input, reducing the computational complexity and the quantity of parameters in the network. Commonly, Max Pooling is adopted. By reducing the size of the feature map, it decreases the computational complexity while preserving significant features and enhancing the model's generalization capability.

Fully connected layer: The features extracted by the convolutional layer and pooling layer are ultimately classified via the fully connected layer. They link each neuron in one layer to each neuron in the subsequent layer, forming the final output, such as classification labels.

Output layer: The output layer typically employs Softmax or Sigmoid functions for classification tasks, outputting the probability of each category. (LearnOpenCV, n.d.)

Owing to the demonstrated efficacy of CNN in image-related tasks, we endeavored to utilize it for handwritten character recognition tasks in this assignment and attained favorable outcomes.

3.2.2 Strengths and Weakness

The strength of CNN lies in its outstanding performance in image recognition tasks, as it can automatically extract features at different levels by constructing a multi-layer network structure, utilizing convolutional kernels, and sharing parameters between different regions, thereby accelerating computation speed. It exhibits excellent translational invariance, can handle various positions and poses of objects in images, and has excellent robustness to noise and changes in input data.

And its weakness is the lack of robustness to rotation and scaling, and it requires a large amount of high-quality data to learn all parameters. This makes CNN have high requirements for datasets, long training times, and high demands on computing resources. In addition, the training and prediction processes within CNN are often difficult to explain, like a 'black box'. It is not easy to directly understand how the model produces predictive results.

3.2.3 Architecture and Hyperparameters

Our model has two convolutional layers which have 32 and 64 3*3 convolutional kernels respectively, and two 2*2 pooling layers, for feature extraction. And then for classification, dropout is used to prevent overfitting after flatten, and finally a fully connected layer is entered, where the softmax activation function is used to classify 62 classes.

The detailed architecture of our CNN model in this assignment is shown in Appendix.

Next for tuning hyperparameters, we choose to tune these hyperparameters: batch_size, epochs, optimizer, and dropout_rate.

Here are the meanings of each hyperparameter, the range of adjustments, and their potential impact on the model:

Batch size: The batch size determines the quantity of samples employed to calculate gradients in each training epoch. Smaller batches update more parameters and weights yet entail longer training periods; larger batch updates are swift, but it might be challenging to break away from local optima, the generalization ability may be slightly worse.

Epochs: The number of times the entire training dataset passes through the model. If the training epochs are limited, the model might be incapable of fully learning complex features, resulting in underfitting. Conversely, a greater number of epochs enables the model to be fully trained but could potentially lead to overfitting.

Optimizer: The optimizer ascertains how to update the weights of the model to minimize the loss function. Different optimizers adopt diverse learning strategies, which can influence the convergence rate and ultimate performance of the model. Adam and Nadam are commonly favored selections.

Dropout_Rate: A higher dropout rate contributes to the prevention of overfitting but might compromise some learning speed and accuracy. A lower dropout rate retains more neurons, which can accelerate the learning speed but might result in overfitting.

These are parameters that may have a significant impact on the performance of the model. Finding the optimal combination of these parameters through GridSearch will significantly improve the accuracy of the model results.

3.3 Model3 MLA

3.3.1 Theory

A Multilayer Perceptron (MLP) is a typical feedforward neural network composed of an input layer, multiple hidden layers, and an output layer. The connections between each layer are fully connected, which is a primary reason for the longer training time associated with this architecture. Each training epoch consists of two stages: the forward pass and the backward pass. During the forward pass, the model predicts the outputs for the training samples, and a loss function is used to calculate the error at the output layer.In the backward pass, the MLP updates the weights of each layer based on the errors computed at the output layer. This process is repeated for all layers until all weights are adjusted, completing one epoch. Throughout both the forward and backward passes, activation functions are employed to introduce non-linearity into the model. Common activation functions include tanh, ReLU, and sigmoid. It is important to note that using the sigmoid activation function can lead to the vanishing gradient problem. This issue arises because the output range of the sigmoid function is between 0 and 1. Consequently, when some error terms are close to 0, the gradients propagated back through the network may become very small, hindering the ability of the earlier layers to learn effectively. One potential solution to this problem is to use the softmax activation function in the output layer for multi-class classification tasks, as it can help to maintain the relative scaling of the outputs. I chose this model because it is easy to implement and it was covered in the Tutorial material.

3.3.2 Strengths and weaknesses

One of its key strengths is its simplicity. The architecture consists of an input layer, a series of hidden layers, and an output layer, which makes it relatively straightforward to implement. Another advantage is that MLP can automatically detect patterns and features from input data without requiring manual feature engineering, making it a powerful tool for complex tasks.

However, MLP also has some notable weaknesses. The first is its computational inefficiency; due to the fully connected nature of the network, it requires a long time to train, particularly when dealing with large datasets and each layer has many neurons. In this project, MLP took the longest time to train compared to other models. Secondly, MLPs, like many deep learning models, suffer from interpretability issues. Unlike simpler models such as linear regression or k-NN, it’s difficult to understand the specific reasons behind its decisions, making it less transparent.

3.3.3 Architecture and hyperparameters
My MLP model consists of an input layer, two hidden layers, and an output layer. Both hidden layers utilize the same sigmoid activation function. The output layer employs the softmax activation function to handle multi-class classification tasks effectively.

I focused on tuning four key hyperparameters in my MLP model. The first two hyperparameters are the number of neurons in the first and second hidden layers. The possible values for these parameters are:

- First Hidden Layer Neurons: "n_units_1": [500, 784]
- Second Hidden Layer Neurons: "n_units_2": [784 // 2, 784 // 3, 784 // 4]

This choice reflects the guideline that the number of neurons typically decreases from the input layer to the output layer. This strategy helps the model learn complex patterns in the data while reducing the risk of overfitting. More neurons may take longer to train.

The third hyperparameter is the activation function, which introduces non-linearity into the model, enabling it to capture intricate relationships within the input features. The potential activation functions I considered are: "activation": ["relu", "sigmoid", "tanh"]. Each activation function has distinct properties that make it suitable for specific scenarios. ReLU (Rectified Linear Unit) is favored for its computational efficiency and ability to mitigate the vanishing gradient problem, allowing for faster training in deep networks and enabling the model to learn complex patterns effectively. tanh (Hyperbolic Tangent) outputs values between -1 and 1, making it zero-centered, which can lead to faster convergence during training. sigmoid is a traditional choice, although it can suffer from the vanishing gradient problem in deep networks, I still decide to add it.

The final hyperparameter is the optimizer, which guides the model in minimizing the loss function during training. The possible optimizers I evaluated are:"optimizer": ["adam", "sgd"]. Adam is often preferred due to its adaptive learning rate, which can enhance convergence speed, while SGD is known for its simplicity and effectiveness in various scenarios. Changing it may change the efficiency of training.

To tune these hyperparameters, I employed a 3-fold cross-validation combined with grid search. This method ensures that each combination of hyperparameters is evaluated across multiple subsets of the training data, providing a robust assessment of their performance and aiding in the selection of the most effective configuration.

4. Experimental Results

4.1 Experimental setting

In this experiment, we use the dataset EMNIST, the detail of it we have mentioned in 2.1, we use the whole test set of it to get the result. We run our models: LSTM, CNN, and MLP for comparing the best model. We choose colab as our environment, the visualized GPU we choose is A100. The hardware is a computer produced by Lenovo, R9000. We import sklearn and tensorflow for building models, and processing dataset. We use pandas and numpy for manipulating data. We import scikeras for initializing the LSTM classifier. We also import matplotlib for comparing the result and visualization.

4.2 Result

4.2.1 Hyperparameter

For LSTM, n_units of it is 150, dropout_rate is 0.2, and optimizer is adam.

For CNN, the batch_size is 32, dropout_rate is 0.3, epochs is 10,and optimizer is adam.

For MLP, n_units_1 is 784, n_units_2 is 392,optimizer is 'adam', activation is 'tanh', learning_rate is 5e-2, and epochs is 15. We show the plot of it in appendix, Figure 10.

4.2.2 Performance and discussion

We can find in this plot that the accuracy of each model is closed, but there still are a few differences.

The highest accuracy is LSTM, which is 0.8363. CNN is 0.8291, and the best MLP is 0.8174.

We also compare precision, recall, F1 score, of each model. For LSTM, Precision: 0.7360 Recall: 0.7016 F1 Score: 0.7030. For CNN, Precision: 0.7199 Recall: 0.6833 F1 Score: 0.6876. And for MLP, Accuracy: 0.8036 Precision: 0.6872 Recall: 0.6751 F1 Score: 0.6720. We show them as a histogram. It seems like LSTM is the best choice for getting high accuracy. Moreover, other metrics in the figure below of LSTM are also the highest.

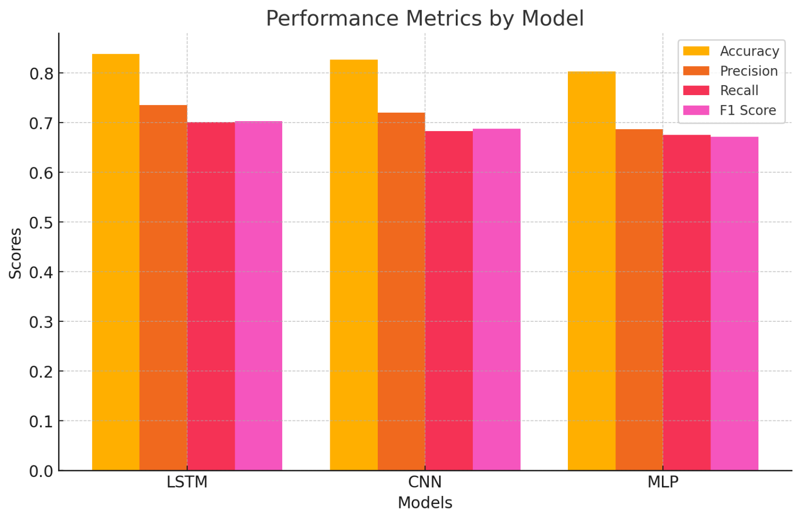


Figure 1. performance of each model.

However, we can't not just compare the metrics such as accuracy, we also need to consider the time cost for each model to justify the performance of them. We find that for LSTM, Training Time: 176.85 seconds Inference Time: 2.01 seconds. For CNN, Training Time: 77.60 seconds Inference Time: 1.36 seconds. And for MLP, Training Time: 80.88 seconds Inference Time: 1.09 seconds. We can find that LSTM has the highest time cost for both training time and inference time, much higher than others, while the value of CNN and MLP are quite close to each other.

So if we want high efficiency, LSTM maybe is not a good choice, CNN is better than it.

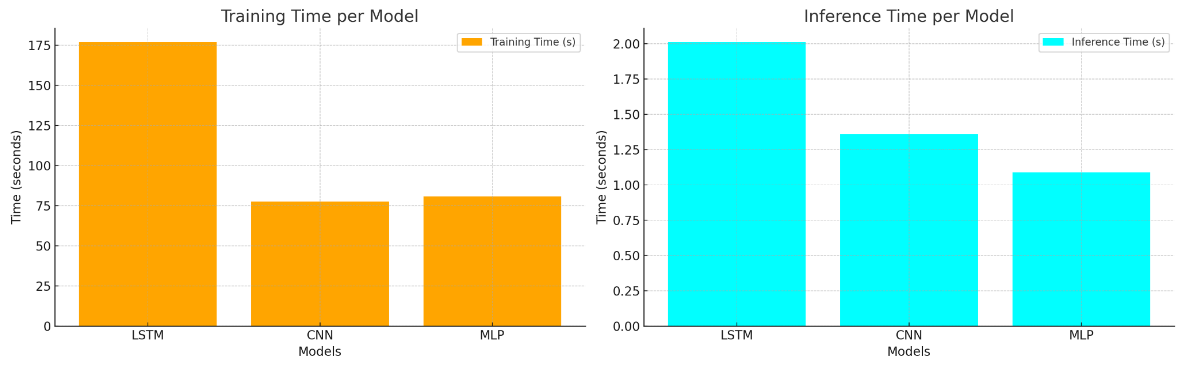


Figure 2 Training time and inference time of each models

The result above shows that the structure of LSTM made its performance good on the large, complex data set. However, because LSTM is not conventionally decided for non-sequential data, it costs a high resource for computation to compensate.

5. Conclusion

5.1 Main finding

In this project, we find that LSTM and CNN are suitable for this job. LSTM gets the highest accuracy, because LSTM can capture temporal dependencies and sequences in data, and it can deal with high-dimensional data very well. Although this model is usually used for natural language processing, it's also a good choice for EMNIST by treating pixel rows as sequences, allowing it to capture some spatial dependencies across the sequence of pixel rows. The next one is CNN. The accuracy of it is very close to LSTM . CNN is adept at extracting local and hierarchical features from the images, leading to higher accuracy and efficiency in image classification tasks. Moreover, we find that if we want to find a better model, we need to spend more time training it, because we need to set more hyperparameters. Moreover, the best model LSTM needs the highest training and inference time. So we have to consider the balance of efficiency and effectiveness of model

5.2 Limited

The model MLP is fully connected, as a result, when this model deals with large and complex dataset like EMNIST, it also has a low efficiency problem, which makes it have a lower performance than others. LSTM and CNN are also not perfect. For LSTM, because we don't use sequential data, the performance of it can't reach the best and need a very long time for training. CNN also needs a complex method for tuning and it needs significant computational resources compared to other models.

5.3 Methods

In this task, we use methods such as data normalization in data preprocessing step, and we tune parameters of CNN one by one in order to save the run time(Maybe it causes the comparative low performance of it). We also use a plot method for comparison of each model.

5.4 Future work

We want to build a complex model combining LSTM and CNN. In our plan, we want to use CNN for feature extraction at first, then we will use LSTM to find the relationship between the features found by CNN. We can use this model to find spatial and sequential relationships between data.

References

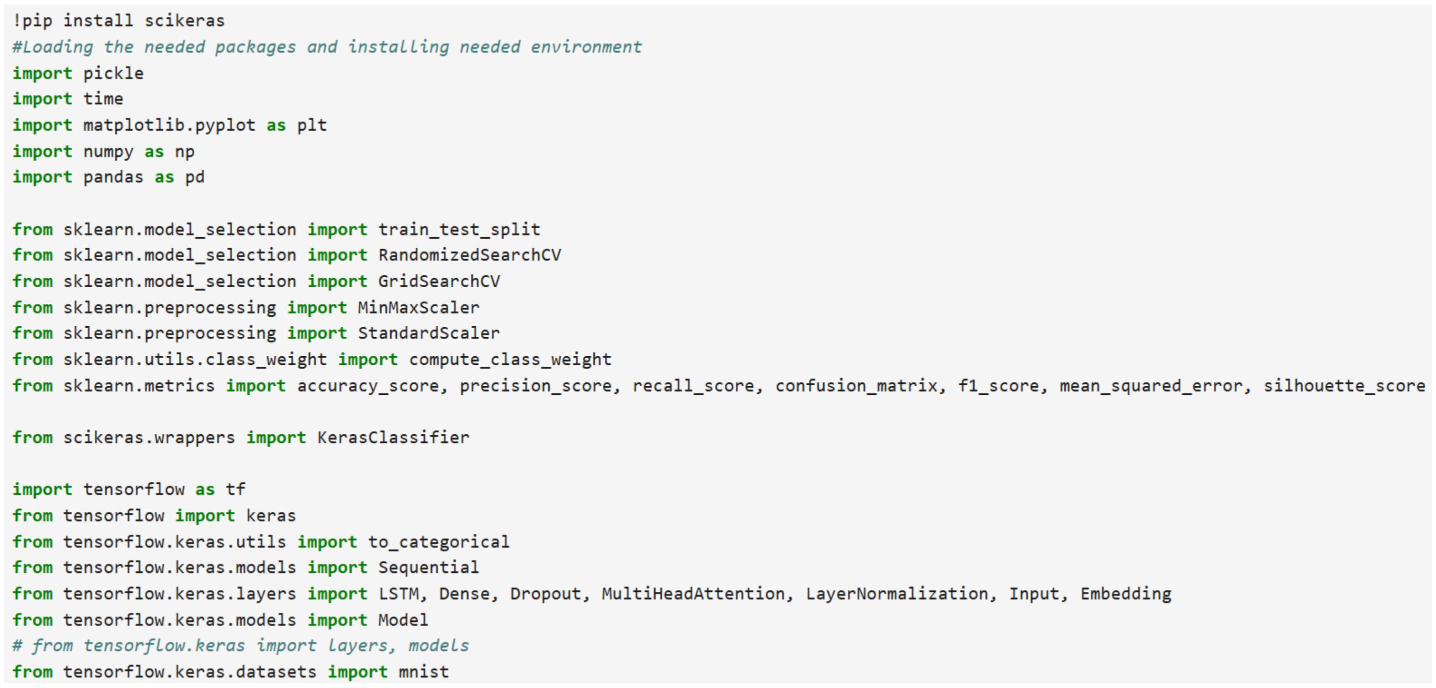
1. Pryor TA, Gardner RM, Clayton RD, Warner HR. The HELP system. J Med Sys. 1983;7:87-101.
2. Gardner RM, Golubjatnikov OK, Laub RM, Jacobson JT, Evans RS. Computer-critiqued blood ordering using the HELP system. Comput Biomed Res 1990;23:514-28.
3. Ranjan, G & KumarA&Radhika, S. (2019). K-Nearest Neighbors and Grid Search CV Based Real Time Fault Monitoring System for Industries. 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), 1-5. Doi: https://doi.org/10.1109/I2CT45611.2019.9033691
4. Anil Chandra Naidu Matcha. (2022). How to easily do Handwriting Recognition using Machine Learning. Retrieved October 11, 2024, from https://nanonets.com/blog/handwritten-character-recognition/
5. Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved October 11, 2024, from https://arxiv.org/abs/1702.05373
6. Zoumana Keita. (2023). An Introduction to Convolutional Neural Networks (CNNs). Retrieved October 12, 2024, from https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns
7. LearnOpenCV. n.d. Convolutional Neural Network (CNN): A Complete Guide. Retrieved October 12, 2024, from https://learnopencv.com/understanding-convolutional-neural-networks-cnn/
8. Brownlee, J. (August 5, 2019). How to Scale Data for Long Short-Term Memory Networks in Python. Machinelearningmastery.https://machinelearningmastery.com/how-to-scale-data-for-long-short-term-memory-networks-in-python/
9. Khan, F., & Qamar, A. M. (2020). A review on long short-term memory (LSTM). IEEE 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). IEEE. https://doi.org/10.1109/ic-ETITE47903.2020.9182390
10. Wikipedia contributors. (n.d.). One-hot. Wikipedia. Retrieved 2024, October 9 from:

    https://en.wikipedia.org/wiki/One-hot

11. Zhao, Z., Chen, W., Wu, X., Chen, P. C. Y., & Liu, J. (2017). LSTM network: a deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2), Article e2016.0208. https://doi.org/10.1049/iet-its.2016.0208

Appendix

In this assignment, we mainly utilized Google's Colab to implement, train, and fine tune our model. By using the Colab platform, we can directly write and run code in the browser without configuring the local environment. Here is the link to colab: https://colab.google/
We also utilized the Jupyter notebook in the Windows system, which can be used by installing Anaconda 3. Here is the installation guide: https://docs.anaconda.com/anaconda/install/windows/
We choose to use Python 3 and A100 GPU in colab, and the necessary packages and libraries are shown below:

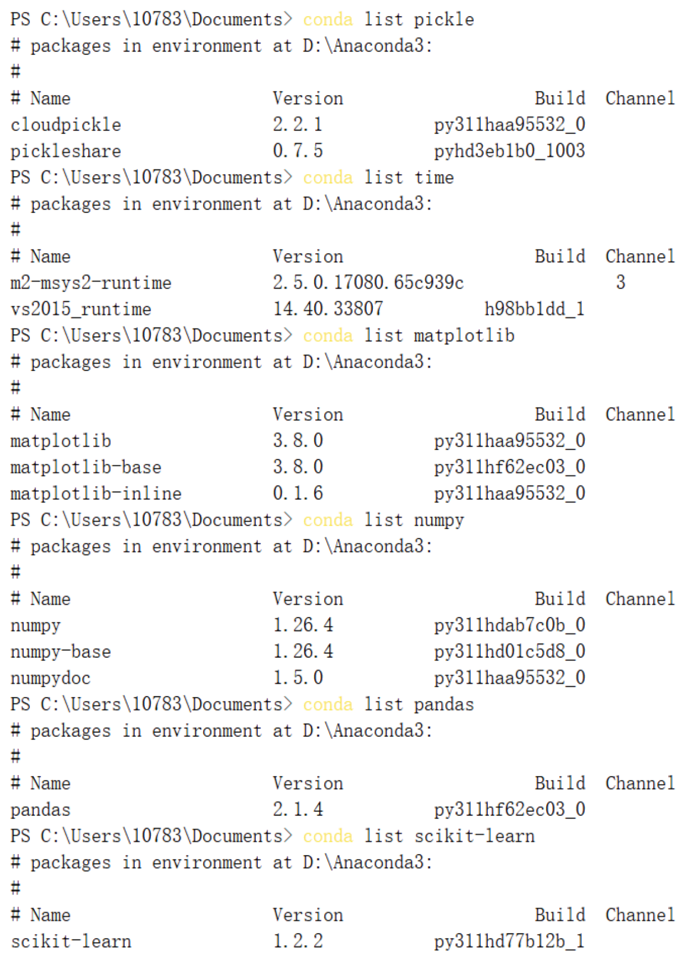


```
!pip install scikeras
#Loading the needed packages and installing needed environment
import pickle
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, mean_squared_error, silhouette_score

from scikeras.wrappers import KerasClassifier

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, MultiHeadAttention, LayerNormalization, Input, Embedding
from tensorflow.keras.models import Model
# from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
```

Figure3. List of libraries and packages

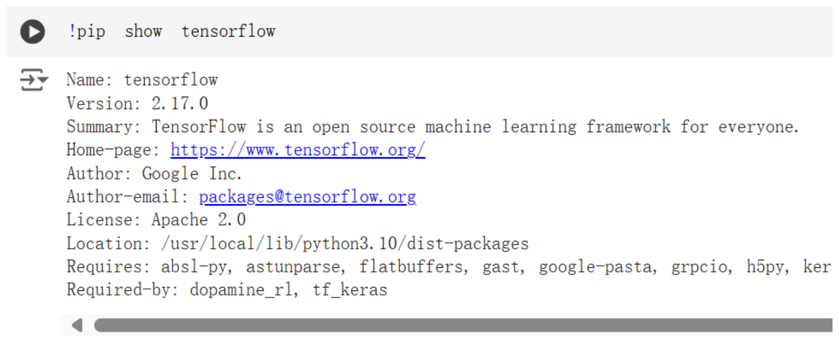
If a library is found to be missing during runtime, just simply use the command '!pip install' to install it, which is also one of the benefits of choosing to use colab.

And the version of necessary libraries and packages are shown below:



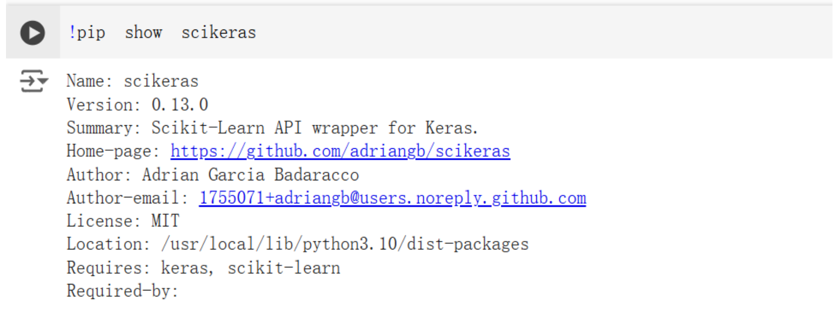
```
PS C:\Users\10783\Documents> conda list pickle
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
cloudpickle               2.2.1           py311haa95532_0
pickleshare               0.7.5           pyhd3eb1b0_1003
PS C:\Users\10783\Documents> conda list time
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
m2-msys2-runtime          2.5.0.17080.65c939c              3
vs2015_runtime            14.40.33807          h98bb1dd_1
PS C:\Users\10783\Documents> conda list matplotlib
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
matplotlib                3.8.0           py311haa95532_0
matplotlib-base           3.8.0           py311hf62ec03_0
matplotlib-inline         0.1.6           py311haa95532_0
PS C:\Users\10783\Documents> conda list numpy
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
numpy                     1.26.4          py311hdab7c0b_0
numpy-base                1.26.4          py311hd01c5d8_0
numpydoc                  1.5.0           py311haa95532_0
PS C:\Users\10783\Documents> conda list pandas
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
pandas                    2.1.4           py311hf62ec03_0
PS C:\Users\10783\Documents> conda list scikit-learn
# packages in environment at D:\Anaconda3:
#
# Name                    Version                   Build  Channel
scikit-learn              1.2.2           py311hd77b12b_1
```

Figure 4. version of necessary libraries and packages(1)



```
!pip show tensorflow
```

```
Name: tensorflow
Version: 2.17.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.10/dist-packages
Requires: absl-py, astunparse, flatbuffers, gast, google-pasta, grpcio, h5py, ker
Required-by: dopamine_rl, tf_keras
```



```
!pip show scikeras
```

```
Name: scikeras
Version: 0.13.0
Summary: Scikit-Learn API wrapper for Keras.
Home-page: https://github.com/adriangb/scikeras
Author: Adrian Garcia Badaracco
Author-email: 1755071+adriangb@users.noreply.github.com
License: MIT
Location: /usr/local/lib/python3.10/dist-packages
Requires: keras, scikit-learn
Required-by:
```

Figure 5. version of necessary libraries and packages(2)

Images that used to showcase the previous content in detail are shown below:

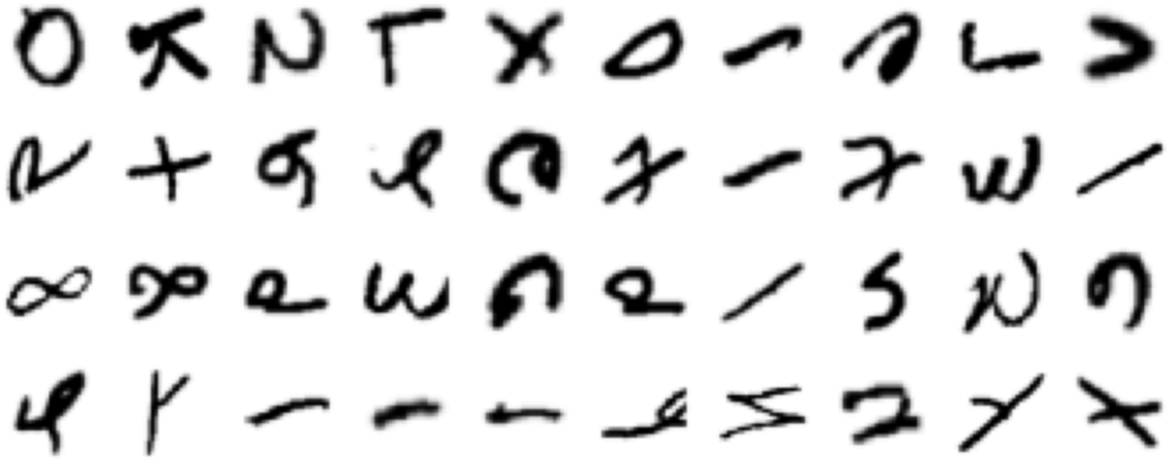
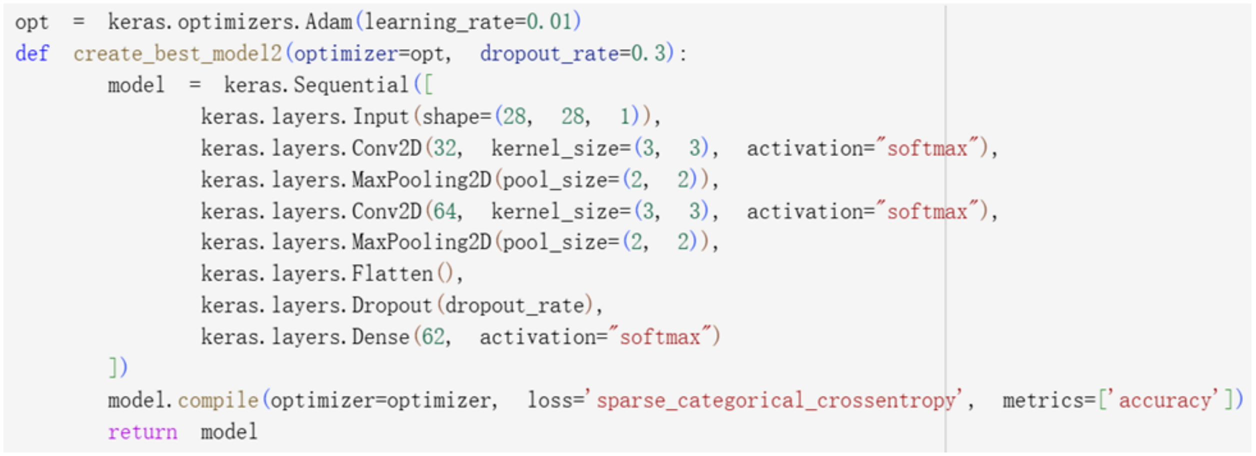
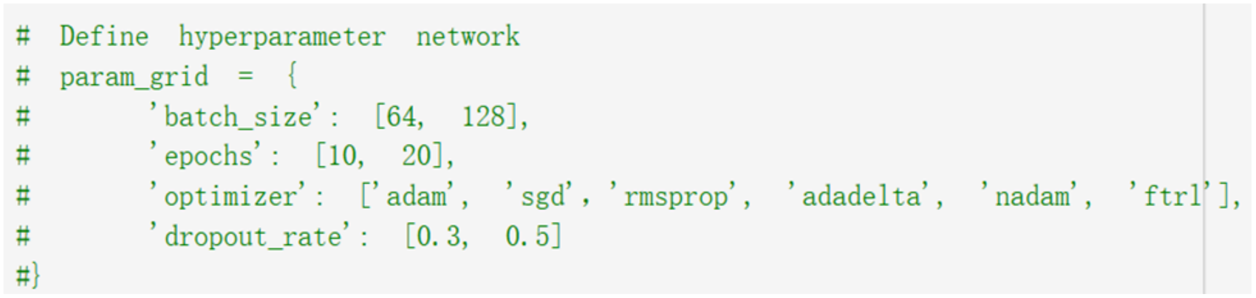


Figure 6. Handwritten Characters in EMNIST



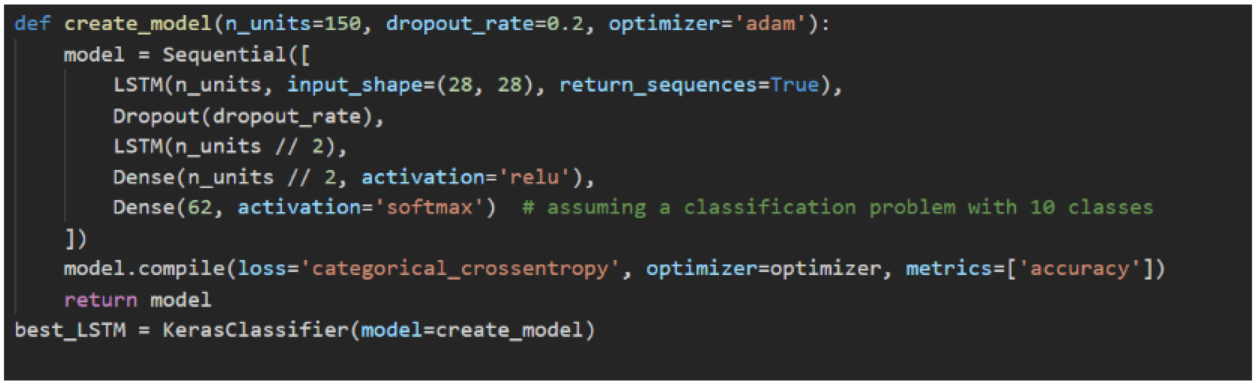
```
opt = keras.optimizers.Adam(learning_rate=0.01)
def create_best_model2(optimizer=opt, dropout_rate=0.3):
        model = keras.Sequential([
                keras.layers.Input(shape=(28, 28, 1)),
                keras.layers.Conv2D(32, kernel_size=(3, 3), activation="softmax"),
                keras.layers.MaxPooling2D(pool_size=(2, 2)),
                keras.layers.Conv2D(64, kernel_size=(3, 3), activation="softmax"),
                keras.layers.MaxPooling2D(pool_size=(2, 2)),
                keras.layers.Flatten(),
                keras.layers.Dropout(dropout_rate),
                keras.layers.Dense(62, activation="softmax")
        ])
        model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        return model
```

Figure 7. Architecture of CNN Model



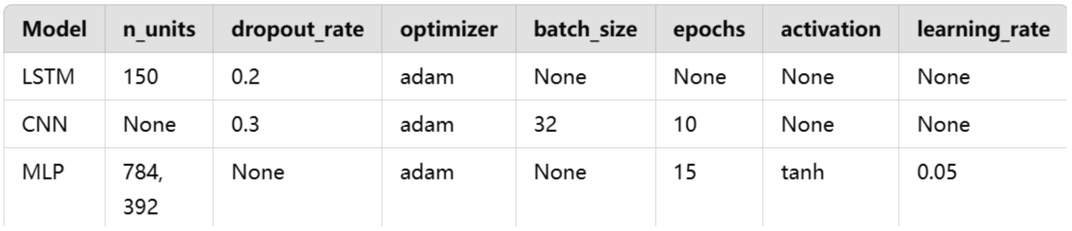
```
# Define hyperparameter network
# param_grid = {
#       'batch_size': [64, 128],
#       'epochs': [10, 20],
#       'optimizer': ['adam', 'sgd', 'rmsprop', 'adadelta', 'nadam', 'ftrl'],
#       'dropout_rate': [0.3, 0.5]
#}
```

Figure 8. Tuning Hyperparameters of CNN Model



```
def create_model(n_units=150, dropout_rate=0.2, optimizer='adam'):
    model = Sequential([
        LSTM(n_units, input_shape=(28, 28), return_sequences=True),
        Dropout(dropout_rate),
        LSTM(n_units // 2),
        Dense(n_units // 2, activation='relu'),
        Dense(62, activation='softmax')  # assuming a classification problem with 10 classes
    ])
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
best_LSTM = KerasClassifier(model=create_model)
```

Figure 9. The architecture of LSTM model



Model	n_units	dropout_rate	optimizer	batch_size	epochs	activation	learning_rate
LSTM	150	0.2	adam	None	None	None	None
CNN	None	0.3	adam	32	10	None	None
MLP	784, 392	None	adam	None	15	tanh	0.05

Figure 10. The hyperparameters of each model