COMP5318 Machine Learning Report

Dongsheng Han, Wei Han, Haochen Zhang

May 21, 2023

Abstract

In this study, we investigate handwritten character recognition using the EMNIST dataset by implementing three machine learning methods: Convolutional Neural Networks (CNN), Support Vector Machines (SVM), and Long Short-Term Memory (LSTM) networks. In order to optimize their performance, we choose different approaches, adopting strategies that best suit their individual characteristics. Through experiments, we evaluate and compare the models' performance using accuracy metrics, providing meaningful insights into the effectiveness of different machine learning models for handwritten character recognition. Finally, the accuracy of the optimal model prediction results of CNN, LSTM, and SVM are 0.854 ,0.852 and 0.886.

1 Introduction

1.1 Project scope

Handwritten character recognition is an important task related to many real-world problems, such as mail recognition and document scanning. In this study, we process well-known EMNIST dataset, which has a large number of labeled images of 52 handwritten letters and 10 numbers. In order to test and compare different machine learning methods with how much they can recognize handwritten characters with these three popular models: Convolutional Neural Networks (CNN), Support Vector Machines (SVM), and Long Short-Term Memory (LSTM) networks. To optimize each model's performance, we focus on selecting suitable methods, such as applying pre-processing techniques like data normalization, and tuning hyperparameters using the grid search algorithm. Finally, we evaluate the models using classification metrics, such as accuracy, precision, and recall, to provide a comprehensive understanding of their performance.

By comparing the performance of CNN, SVM, and LSTM networks in recognizing handwritten characters, we aim to provide valuable insights into the strengths and weaknesses of each method, and help researchers working on similar problems in the field of character recognition or other real-world applications.

1.2 Dataset background

The EMNIST (Extended Modified National Institute of Standards and Technology) dataset is an expanded version of the well-known MNIST dataset, which is used extensively in the field of machine learning and computer vision for tasks like handwriting recognition [CATvS17]. It features a set of 28x28 pixel grayscale images of handwritten characters and digits.

In this project, we used the smaller dataset contains two files with two keys, the data in `emnist_train.pkl` would typically be used to train a model, while the data in `emnist_test.pkl` would be used to evaluate the model's performance on data after training.

emnist_train.pkl

data: A numpy array of shape (100,000, 1, 28, 28). It contains 100,000 images, each of which is a grayscale representation of a 28x28 pixel handwritten character or digit. The number '1' in the shape of the array represents the single color channel, as these are grayscale images.

labels: A numpy array of shape (100,000,). This array holds the corresponding labels for each image in the data array. These labels are integers ranging from 0 to 61, representing 62 distinct classes in the EMNIST dataset's 'by_class' categorization. These classes correspond to the alphanumeric characters (A-Z, a-z, and 0-9).

emnist_test.pkl

data: A numpy array of shape (20,000, 1, 28, 28). This array contains 20,000 grayscale images, each a 28x28 pixel handwritten character or digit, just like the training data.

labels: A numpy array of shape (20,000,), holding the class labels for each test image. These labels are also integers from 0 to 61, representing the same 62 classes as the training data.

2 Previous Work

In the field of handwritten character recognition, many studies have adopted image rotation as a pre-processing step for the EMNIST dataset. One introductory example is the work by Cohen et al. (2017), who performed an in-depth analysis on it.

The authors explained that the images in the EMNIST dataset were initially stored in a format where the characters were rotated 90 degrees counter-clockwise and flipped. This format was chosen to maintain compatibility with the original MNIST dataset.[CATvS17] However, in order to properly visualize and analyze the images, it was necessary to apply a rotation and flip operation to correct the orientation of the characters. Following this approach, we can ensure that the characters are correctly oriented before training machine learning models. By doing so, we may have been able to improve the performance of their models and provide more accurate character recognition results.

3 Data Pre-processing

3.1 Normalisation

Normalize the data by scaling it to the range of 0-1 is necessary in the project to bring all the features into a consistent scale. The variable x_train_full is cast to the data type 'float32' to ensure numerical precision. Then, each element in x_train_full is divided by 255 to scale the values between 0 and 1. This normalization step is often performed to ensure consistent and efficient training of machine learning models.

3.2 Rotate images

As we mentioned in previous work, image rotation is commonly used as a pre-processing step to improve the accuracy of character recognition in their models.

```
# Define a rotate() function
def rotate(image):
    image = np.fliplr(image)
    image = np.rot90(image)
    return image
```

Therefore, we defined a function rotate(), which flips the image horizontally and rotates it by 90 degrees as we explained in previous work. This step ensures that the images are correctly oriented for further processing. We then applied this function to each sample in the training and testing dataset.

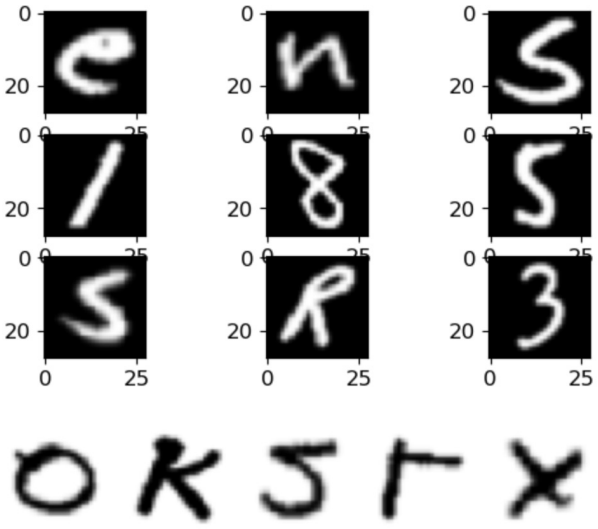

Figure 1: Handwritten data sample after rotation

3.3 Split dataset

The training data is split into training and validation sets. The train_test_split function from the scikit-learn library is used for this purpose. The x_train_full and y_train_full datasets are divided into a training set (x_train and y_train) and a validation set (x_valid and y_valid). The parameter train_size=0.9 specifies that the training set should contain 90% of the original data, while the remaining 10% will be allocated to the validation set. This split allows for model evaluation and tuning during the training process.

4 Convolutional Neural Networks

4.1 Methodology

Convolutional Neural Networks (CNNs) can offer advantages for our EMNIST dataset as it's very well-suited for the image recognition. CNN is one of the best methods to perform classification on image data. It mimics the visual cortex of the human brain, which can detect complex features from the image to recognize an image [Aea17]. The same principle is applied in CNN, where different types of filters are applied over the image to extract feature images, and prediction is performed using a neural network.

CNNs have fully connected layers and softmax activation to perform classification for images extracted features. During the training, network's parameters are processed through back-propagation, optimizing the cross-entropy loss function. This allows us to generalize handwritten letters by learning its representations and capturing underlying features in the EMNIST dataset.

4.2 Experimental Setting

The experiments were conducted using the EMNIST dataset, which contains labeled images of 52 handwritten letters and 10 numbers, the images size is 28x28, and there are 100'000 images in training set and 20'000 in test set. We implemented the CNN model using Python and popular libraries: TensorFlow and Keras. This experiment was done by Dongsheng Han with an Apple M2 Pro MacBook Pro.

The proposed CNN model consists of sequential layers that were determined after a parameter grid search. The input layer is defined with a shape of (28, 28, 1), representing the input image size and the number of color channels, it's 1 in this case since we are processing gray-scale images. The first layer is a convolutional layer with 32 filters and the kernel size is (5, 5). The activation function used is ReLU. Following the first convolutional layer, there is a max-pooling layer that reduces the spatial dimensions of the feature maps by picking the maximum value in each pooling region.

Subsequently, two additional convolutional layers are connected sequentially. The first has 64 filters with a kernel size of (3, 3) and the activation function is ReLU. The second has 128 filters with the same kernel size and activation function. After the convolutional layers, a max-pooling layer is applied.

The model then includes a flatten layer, which transforms the 2-dimensional feature maps into a 1-dimensional representation. This is followed by two dense layers. The first dense layer has 100 units and the activation function is ReLU. A dropout layer with a rate of 0.5 is inserted between the two dense layers to randomly drop weights during training, helping to prevent overfitting. The last dense layer has 62 units, representing the number of classes to be classified, and the activation function is softmax.

Overall, the model architecture includes 637,724 trainable parameters. The Adam optimizer is used with the learning rate and other hyperparameters set to their best values. The loss function is categorical cross-entropy, suitable for multi-class classification, and the accuracy metric is used for evaluation.

4.3 Implementation

After rotating the images in the EMNIST dataset, we proceeded with the CNN implementation. Firstly, we applied one-hot encoding to the labels. One-hot encoding is very effective as it helps transform the categorical labels into a binary format, where it could be easier for computer to process and understand the class.

In order to find the best parameter, we used `GridSearchCV` function and defined a function `create_cnn_model` that creates the CNN model basic architecture. The function takes parameters such as `optimizer`, `kernel_size`, and `num_filters` as inputs and returns the compiled model. After that we wrapped the `create_cnn_model` function with `KerasClassifier`. This conversion allows us to use the Keras model within the scikit-learn pipeline.

Then we set up a dictionary parameters containing hyperparameter values to be searched:

- `optimizer = ['adam', 'rmsprop', 'SGD']`
- `kernel_size = [(3, 3), (5, 5)]`
- `num_filters = [32, 64]`
- `batch_size = [64, 128]`
- `epochs = [10, 20]`

The `CV` parameter was set to 2 as it specifies two cross-validation folds, and `verbose = 3` controls the verbosity of the search process. After a very long time grid search is completed, we printed the best parameters found:

CNN - Grid Search	**Best Params**
batch size	128
epoch	10
kernel size	(5, 5)
number of filters	32
optimizer	adam

Table 1: CNN Best Params Results

Next, we can finally create the best CNN model using the best parameters obtained from the grid search. The model architecture is defined using the `Sequential` API from `tensorflow.keras`.

```
from tensorflow.keras.optimizers import Adam

model_cnn = keras.Sequential([
    keras.Input(shape=(28, 28, 1)),
    keras.layers.Conv2D(32, kernel_size=(5, 5),
        activation="relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(64, kernel_size=(3, 3),
        activation="relu"),
    keras.layers.Conv2D(128, kernel_size=(3, 3),
        activation="relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.4),
    keras.layers.Dense(62, activation="softmax"),
])

model_cnn.compile(optimizer='adam',
                  loss="categorical_crossentropy",
                  metrics=["accuracy"])
# Best model fit
history = model_cnn.fit(x_train_cnn, y_train_cnn,
                        batch_size=128,
                        epochs=10,
```

```
26                                  validation_data=(
27                                      x_valid_cnn, y_valid_cnn)
28                                  )
```

4.4 Experimental Results

The training process took approximately 130 seconds, accelerated using GPU mode. Finally, we evaluated the model's performance on the test dataset, achieving an overall accuracy of 0.8549.

Moreover, we plotted the training accuracy and validation accuracy during the training process to visualize the model's performance over time. This allowed us to gain insights into the model's generalization capability and to ensure it was not overfitting the training data.

To demonstrate the correctness and efficiency of our CNN implementation, we present our experimental results using various visual representations such as charts, graphs, and tables. We randomly selected three test samples from the dataset and compared their predicted results with the true labels. The predicted results were 9, 37, and 3, which correspond to the characters "9", "b", and "3", respectively. These predictions aligned with the true labels, providing evidence of the model's accuracy in recognizing handwritten characters.

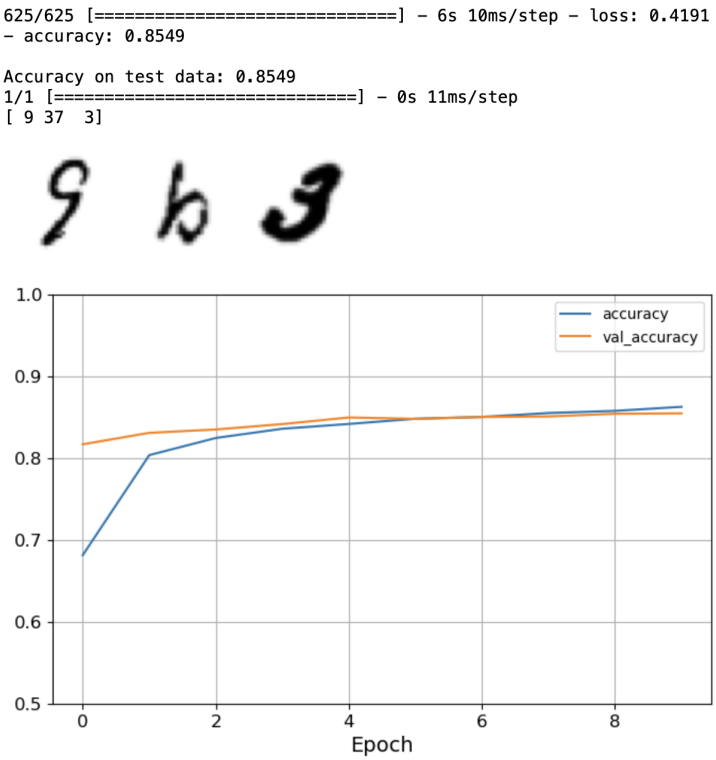

Figure 2: Accuracy on test set and Training-Validation Accuracy

4.5 Reflection

As we all know, CNNs have excellent performance in image classification tasks, and that's the reason why we choose it to perform this task. Despite our efforts to improve the accuracy of our model, we encountered limitations that we cannot get further performance improvements. We have tried several approaches:

- Data Augmentation: Use `ImageDataGenerator` to apply some random transformations to the input images, such as rotating 10 degrees or randomly shift left 10 percent, therefore it can provide wider range of training examples, increasing its ability to learn robust and generalized features.

```
data_gen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode="nearest")
```

- Early Stopping: It is a technique used during our previous model training to prevent overfitting and improve generalization. It monitors a validation metric and stops the training early if the metric stops improving. But finally, as the best parameter combination was found at 10 epochs, which is a relatively low number of training iterations, the model did not require the use of early stopping.

```
# Stop at 3 consecutive steps
# without improvement in the validation loss.
early_stopping = EarlyStopping(
        monitor='val_loss', patience=3)
```

- VGG16 pre-trained model: It is originally trained on RGB images, which have three color channels (red, green, and blue). We tried to use this well-defined model to improve our performace. However, it becomes a challenge when adapting it to handle EMINST single-channel grayscale images due to the channel mismatch. In this case, duplicating the single channel three times may not fully utilize the capabilities of the VGG16 model, which is primarily designed to extract features from color images. Therefore, we finally discard this method.

As we reflect on our work, we known that CNNs usually have excellent performance in image classification tasks, which motivated our choice of using this approach for the EMNIST dataset. We have achieved respectable accuracy with 88% on a dataset of 47 handwritten characters and 99% on the 10-digit dataset. However, given the increased complexity and variability of the 62-character EMNIST dataset, further enhancements in accuracy may be more difficult a lot. It requires further exploration and investigation into CNNs structure.

5 Long Short-Term Memory networks

5.1 Methodology

Our dataset is about handwritten characters. A complete character is difficult to represent with a small image. So when the model recognizes a character, it needs to analyze all connected blocks showing the whole character. This means that using a neural network model with memory may have good predictions. And we don't know how many pieces a character needs.

If our blocks are too small or the characters are too large, the neural network will forget the previous input. LSTM can perfectly meet these requirements.

Because it is a classification method suitable for sequential data. LSTM is a recurrent neural network that can capture long-term dependencies in sequence data through memory cells and gating mechanisms.

For the input of LSTM, we have performed data preprocessing, because each data has 62 possibilities, so we have performed "tocategorical" processing on the training set, test set and verification set.

```
# Preprocess the MNIST dataset
y_train_lstm = to_categorical(y_train, num_classes=62)
y_valid_lstm = to_categorical(y_valid, num_classes=62)
y_test_lstm = to_categorical(y_test, num_classes=62)
```

An LSTM model consists of an input layer, a hidden layer, and an output layer, where the hidden layer contains LSTM cells. An LSTM cell consists of three gates: an input gate, a forget gate, and an output gate. The input gate controls whether new inputs enter the LSTM cell, the forget gate controls whether the previous state is forgotten, and the output gate controls the output value. LSTM models can be trained through the backpropagation algorithm to minimize a loss function. During training, we can use the backpropagation algorithm to optimize the cross-entropy loss function to learn the representation and features of handwritten letters in the EMNIST dataset. LSTM can use relu or other activation functions and fully connected layers for classification prediction. During training, we can use batch normalization and dropout techniques to improve the performance and generalization of the model. In this way, LSTMs can capture the sequence features in the EMNIST dataset to achieve accurate classification predictions for handwritten letters.

5.2 Experiment Setting

We implemented the LSTM model using Python and popular libraries: TensorFlow and Keras. This experiment was done by Haochen Zhang with an Apple 2019 MacBook Pro.

LSTM has sequential layers including input layer, LSTM layer, dense layer and output layer. The input layer is (28, 28), which is the images size.

Then, the input data will pass through the LSTM layer. The LSTM layer can rely on the neural network with long-term memory to help make predictions.We experimented with many different LSTM model parameters to achieve excellent prediction accuracy. We try to control four important parameters (units, activation, recurrent activation, dropout) to improve the model.

1. Units affect prediction accuracy and generalization ability.
2. Activation function affects the non-linearity of the model. And its ability to capture non-linear patterns in the data to achieve better predictions.
3. Recurrent activation function will affect the model's ability, to capture long-term dependencies in the data for better predictions.
4. The LSTM layer uses the dropout rate to control the degree of overfitting of the model, which can improve the generalization ability of the model, prevent overfitting of the training data, and obtain better prediction performance.

Next layer is the Dense layer. We added a dense layer with 62 neurons using a softmax activation function. The output of this Dense layer is converted to a probability distribution, where each element represents the model's predicted probability for each class. During the training process, the model will update the weight of the model according to the difference between the real label and the predicted probability, thereby improving the prediction accuracy of the model.

Finally, the data will output a data with a shape of (number of samples, 62) through the output layer.

5.3 Implementation

I also used `optimizer="adam"`, `loss="categorical_cross_entropy"`, `metrics=["accuracy"]` for LSTM model. Because my computer has limited computing power, I set the `batch_size` to 32 and the `epochs` to 10.

We mainly put the work of adjusting parameters on the LSTM layer:

- units = [64,128,256]
- activation = ['relu', 'elu']
- recurrent_activation = ['tanh','sigmoid']
- dropout = [0,0.1,0.2]

After test all the different combination of parameters, we get many model with test accuracy. And according to the comparison of test accuracy, we find the model has the best predictive ability.

LSTM - Grid Search	**Best Params**
units	256
activation	elu
recurrent activation	sigmoid
dropout	0.2

Table 2: LSTM Best Params Results

Next, we create the best LSTM model using the best parameters obtained from the keras LSTM layer. We use the tensorflow model.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Reshape
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping
lstm_model = Sequential([
    Reshape((28, 28),
    input_shape=(28, 28)),
    LSTM(units, activation='elu',
         recurrent_activation='sigmoid',
         dropout=0.2),
    Dense(62, activation='softmax'),
    Reshape((62,))
])
```

```
best_model.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history = best_model.fit(x_train,
                         y_train_lstm,
                          epochs=10,
                          batch_size=32,
                        validation_data=(x_valid, y_valid_lstm))
```

5.4 Experiment Result

The training process takes approximately 5 hours using CPU mode. Finally, the test accuracy of best model is 0.852.

In LSTM ease we similarly plot the training accuracy and validation accuracy during training to visualize the performance of the model over time. It also ensures that the fit of LSTM is within an acceptable range.

In order to better compare with other models, for the prediction of LSTM, the corresponding characters "9", "b", and "3" are also selected as samples, and the final prediction results are also 9, 37, 3. The prediction has also reached Accurate effect.

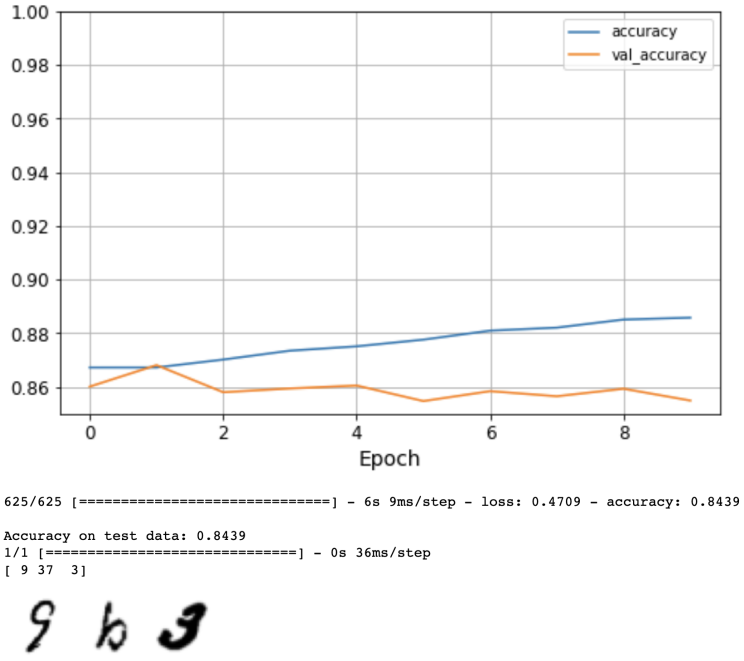

Figure 3: Accuracy on test set and Training-Validation Accuracy

5.5 Comparison

In terms of performance, the prediction result is similar to CNN, but it is much worse than SVM. Maybe the pixel size we choose will also greatly affect the thinking of LSTM. For example, if the pixel is too small, because the proportion of the handwritten part to the overall image is

too small, it will cause a large number of inputs to be 0, which will lead to Long-term memory hardly affects subsequent predictions.

In terms of prediction time and computational complexity, the prediction speed is faster when the number of units in the lstm layer is small. When the number of units increases, the calculation time will increase significantly.

- 64 units: 20s/epochs
- 128units: 40s/epochs
- 256units:80s/epochs

5.6 Reflection

The LSTM model has a long-term memory. I think that each black pixel of each handwritten character is connected together, so the long-term memory may have a better prediction effect.We tried several methods:

- Early stopping: Because the prediction time of LSTM is long, earlystopping is also used in this part to reduce the calculation time.

```
lstm_model = build_lstm_model(units, activation,recurrent
,drop)
# Compile the model
lstm_model.compile(optimizer='adam', loss='categorical_
crossentropy', metrics=['accuracy'])
# Train the model
early_stopping = EarlyStopping(patience=3,
 restore_best_weights=True)
lstm_model.fit(x_train, y_train_lstm, epochs=10,
batch_size=32, validation_data=(x_valid, y_valid_lstm))
# Evaluate the model on the test set
test_loss, test_acc = lstm_model.evaluate(x_test,
y_test_lstm)
print(f"Units: {units}, Activation: {activation},
 Test Loss: {test_loss}, Test Accuracy: {test_acc}")
if test_acc > best_acc:
    best_model = lstm_model
    best_acc = test_acc
```

- Try different activation functions: During the experiment, I tried 'relu', 'elu', two different activation functions, and two recurrent activations of sigmoid' and 'tanh'. I compared the prediction results and selected the most suitable combination for predicting handwritten characters. The Exponential Linear Unit (ELU) is an activation function that can be used in LSTM models.[[Cle16]]

```
units_list = [64, 128, 256]
activation_list = ['relu', 'elu']
recurrent_activation=['tanh','sigmoid']
dropout=[0,0.1,0.2]
best_model = None
```

```
best_acc = 0
```

6 Support Vector Machine

6.1 Methodology

Supporting Vector Machine (SVM) is a binary classification model based on statistical learning theory, which uses margin maximization on the feature space for classification. For linearly separable problems, SVM uses quadratic programming to achieve the maximum classification interval; for nonlinear classification problems, SVM first maps the input space to a high-dimensional space through a suitable kernel function, and then realizes linear separability in the high-dimensional space. The purpose of SVM nonlinear mapping is to seek an optimal hyperplane to maximize the classification interval, and at the same time, achieve the best learning generalization ability and mapping complexity of sample data.[[VSK19]]

Handwritten characters in the EMNIST dataset may have complex shapes and structures, which are difficult for linear classifiers to accurately classify. SVM achieves non-linear classification capabilities by using a kernel function to map features to a high-dimensional space. This enables SVM to handle more complex handwritten character classification problems.

6.2 Experimental Settings

We implemented the SVM model using Python and popular libraries: Sklearn. This experiment was done by Wei Han on Google Colab Platform.

6.2.1 Data Pre-processing

In multi-classification problems, feature engineering is crucial to the performance of SVM algorithms. Selecting and extracting appropriate features can improve the accuracy and robustness of classifiers.

1. Change from matrix to array: Each sample in the original EMNIST dataset is a 28x28 two-dimensional matrix representing pixel values of handwritten characters. First, dimension conversion is required to convert the data format from matrix to one-dimensional array(784=28x28).
2. Split the dataset: We split the dataset into training, validation and testing sets, ensuring an appropriate distribution of digit classes in both sets.
3. PCA analyze: Then apply PCA without reducing the dimensionality and compute the minimum number of dimensions (features) required for preserving 95% of the variance. It shows that 103 features are needed to preserve 95% of the variance. This is a considerable compression and only uses 15% of the original features.

6.2.2 Fine-tuned

It is very important to choose the kernel function, gamma and C in the SVM algorithm, and they will have a significant impact on the performance of the model.

- Kernel selection: If the data has a complex nonlinear distribution, you can choose the radial basis function (RBF Kernel) as the default choice, which can provide good performance in most cases.

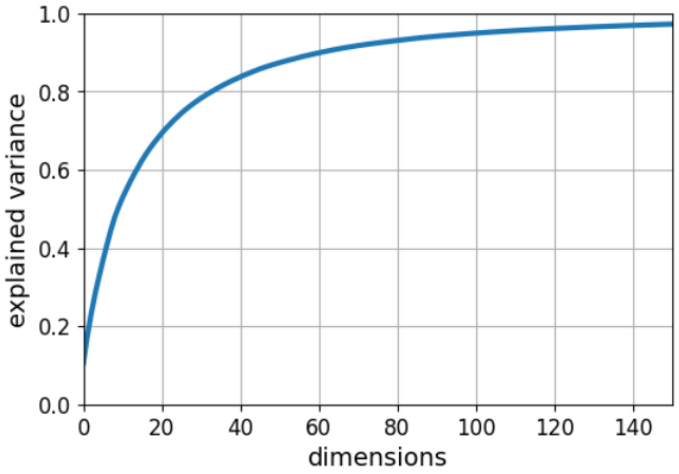

Figure 4: the explained variance vs number of dimensions

- Gamma selection: Gamma is a hyperparameter of the radial basis function that controls how curved the decision boundary is. Smaller values of gamma make the decision boundary smoother, possibly leading to underfitting. Larger gamma values will make the decision boundary more complex and may lead to overfitting.
- C selection: C is the penalty parameter of the SVM algorithm, which controls the degree of penalty for classification errors.A smaller value of C will make the decision boundary smoother, tolerate more misclassification, and may lead to underfitting. Larger values of C will make the decision boundary tighter, possibly leading to overfitting.

In this experiment we choose the grid search technique to evaluate the performance under different parameter combinations, so as to determine the best parameter configuration.

6.2.3 Implementation

In order to find the best parameter, we used the GridSearchCV function. The function takes parameters such as kernel, gamma, and C as inputs and returns the compiled model. Then we set up a dictionary parameters containing hyperparameter values to be searched:

- `kernel = ['rbf', 'linear']`
- `C = [0, 10, 100, 1000], [0, 10, 100, 1000]`
- `Gamma = [1e-3, 1e-4]`

The CV parameter was set to 2 as it specifies two cross-validation folds. After a very long time grid search is completed, we printed the best parameters found:

SVM - Grid Search	**Best Params**
kernel	rbf
gamma	1000
C	0.001

Table 3: SVM Best Params Results

Next, we can finally create the best SVM model using the best parameters obtained from the grid search. The model architecture is defined using the SVC API from *sklearn.svm*.

```
1 # best model fit
```

```
svm = SVC(kernel="rbf", C=1000, gamma=0.001)
svm.fit(x_train_svm , y_train_svm)
```

6.3 Experiment Result

The training process took approximately 3 hours, accelerated using GPU mode. Draw the accuracy heatmap to see the prediction effect of the SVM model under different parameters. Finally, we evaluated the model's performance on the test dataset, achieving an overall accuracy of **0.8857**.

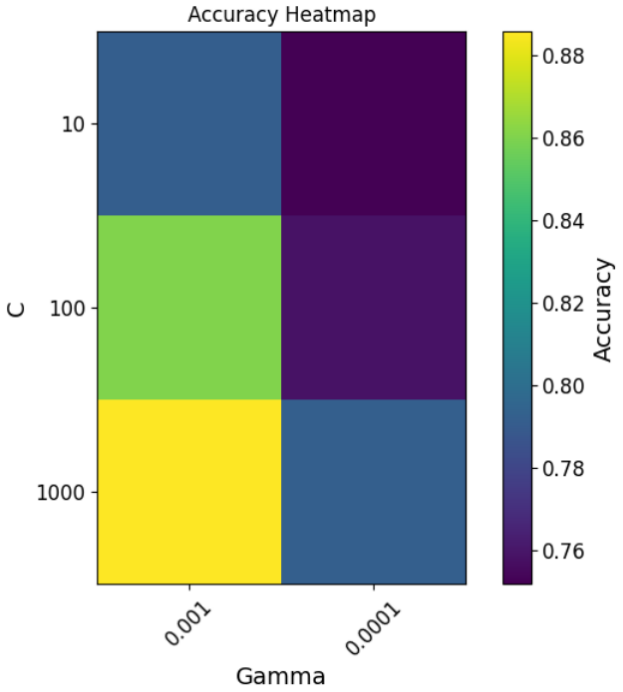

Figure 5: the accuracy heatmap

6.4 Reflection

Reflecting on the experiment, several key observations and insights can be drawn:

1. SVM Performance: SVM achieved competitive classification accuracy on the EMNIST dataset, showcasing its capability in handling complex classification tasks. The ability of SVM to handle both linear and non-linear decision boundaries contributed to its success in recognizing diverse handwritten digits.
2. Hyperparameter Tuning: Proper hyperparameter tuning played a crucial role in maximizing SVM's performance. The regularization parameter (C) and kernel-specific parameters (e.g., gamma for RBF kernel) needed careful adjustment. Overfitting or underfitting could occur if the hyperparameters were not properly tuned. Iterative experimentation and validation were necessary to find the optimal configuration that balanced model complexity and generalization ability.
3. Model Optimization: Further improvements can be explored to enhance SVM's performance on the EMNIST dataset. Feature engineering techniques, such as extracting more informative features from the raw pixel values, could potentially boost the classification

accuracy. Additionally, ensemble methods or cascading SVM with other classifiers might yield better results by leveraging the strengths of different models.

SVM is the slowest model among the three models, but it has the best result. Our experiments demonstrated the effectiveness of SVM in accurately recognizing handwritten digits. We discussed the impact of different kernel functions and hyperparameters on classification performance. The results emphasize the significance of appropriate kernel selection and hyperparameter tuning for achieving high accuracy. SVM remains a promising algorithm for digit classification tasks, and further optimizations can be explored to enhance its performance.

7 Conclusion and Future Work

After conducting experiments and exploring different approaches, we can conclude that using a complex CNN model may be an overkill for the EMNIST dataset. The achieved accuracy of 86% indicates that the model's performance is quite stable, and further improvements are challenging to achieve. To improve the model's performance, we can explore additional hyperparameter combinations, such as incorporating more convolutional layers or trying out more complex CNN architectures.

The prediction accuracy of LSTM is 85.2%. In theory, LSTM should be able to optimize the prediction effect through memory, but neither adjusting the input pixel size nor activation has significantly improved the prediction accuracy, and even some combination of activation and reccurent activation have extremely low prediction accuracy. Therefore, LSTM is not the optimal choice for the prediction of handwritten characters. It may be because there are too many blank parts, which makes it difficult to form useful memories.

The SVM algorithm performs well on smaller datasets, especially if the feature dimensionality is not high. In this experiment, the prediction accuracy of the svm model trained by the optimal parameters reached more than 88%. The EMNIST dataset has more categories and samples than the original MNIST dataset, so for the SVM algorithm, it may require more computing resources and longer training time to process.

Overall, the dataset consists of grayscale images of handwritten characters, and its relatively small size and simplicity may not fully benefit from the complexity of the model. By exploring a wider range of hyperparameter combinations and considering various architectural choices, we can potentially achieve better performance on the EMNIST dataset.

References

[Aea17] Saif Albawi and et al. Convolutional neural networks applied to handwritten bangla numeral recognition. *Journal Name*, 2017.

[CATvS17] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. 2017.

[Cle16] Unterthiner T. Hochreiter S Clevert, D. A. Fast and accurate deep network learning by exponential linear units (elus). 2016.

[VSK19] Greeshma K V and André Sreekumar K. Fashion-mnist classification based on hog feature descriptor using svm. 2019.