THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: COMP5318 Machine Learning and Data Mining

Assignment name: Assignment 2 – Group 66

Tutorial time: Mon 18:00, Tue 18:00 Tutor name: Tahsin Samia, Canh Dinh

DECLARATION

We the undersigned declare that we have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, an, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that failure to comply with the *Academic Dishonesty and Plagiarism in Coursework Policy* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Contribution Form (Each group member contributed equally to this project.)				
Student name	**Student ID**	**Participated**	**Contribution**	**Signature**
Zedong Wen	500334863	Yes	Access to relevant information and build model architecture. Completed most of the code and participated in the preparation of the report.	Zedong Wen
Han Liu	480010528	Yes	Arranged meetings and fully discussed ideas. Completed most parts of the report and participated in code runs.	Han Liu

SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A

Assignment 2 - Group: 66

Tutors: Tahsin Samia, Canh Dinh

Group members:Zedong Wen(zwen8150), Han Liu(hliu5424)

Abstract

As the extended MNIST (EMNIST) dataset was developed, a more challenging benchmark for the handwritten character classification task was appeared. In this study, the main contributions are that high-performance classifiers to classify the 62 labels of the by class subset in EMNIST are proposed. The state-of-the-art work on image recognition is referenced, our proposed models are based on three deep neural network algorithms, AlexNet, VGGNet, and ResNet, which have won the ILSVRC-ImageNet competition champion in recent years. By changing the model architecture or the size of the convolution kernel, three models with improved characteristics of the EMNIST dataset were built for each original model, and the best performing model achieved an accuracy of 87.65% that based on AlexNet algorithm.

Keywords: Extended MNIST (EMNIST); image classification; AlexNet; VGGNet; ResNet

1 Introduction

1.1 What is the problem?

In the past few years, classification has received increasing attention as a popular and challenging area of research in machine learning [1]. Many classification algorithms such as nearest neighbor, decision tree, random forest, neural network, Bayesian network, support vector machine and recurrent and convolutional neural network have been well developed and become reliable techniques in various computer vision tasks such as: objects in tiny images [2] [3], medical image analysis [4][5] and image classification [6][7][8].

Some previous studies give ideas about datasets and classification algorithms. Since different datasets may be suitable only for a specific algorithm serving a specific task, it is important to select a good and diverse set of datasets in order to judge and evaluate the performance of classification algorithms more comprehensively.

More specifically, the most widely known dataset is probably the MNIST dataset [9]. However, a large amount of work is now done on this dataset with a test error rate of less than 1%, which becomes less challenging. In 2017, an extended version of MNIST was released [7], introducing a new dataset: EMNIST, which deals with numbers and upper and lower case letters and has the same structure as the MNIST database. Six different classifications are provided in this dataset. They are ByClass, ByMerge, Balanced, Letters, Digits and MNIST. In this study, we use the EMNIST ByClass dataset and use the classification algorithm to deal with the classification problem(image classification) in this data

1.2 Why is the problem important and overview of the methods?

91% of the information that humans understand about the world comes from vision [10]. Similarly, computer vision has become the foundation of machine recognition. Computer vision is an interdisciplinary comprehensive technology, and its application scope covers all industries of the national economy, such as industry, agriculture, medicine, military affairs, aerospace, meteorology, astronomy, public security, transportation, security, scientific research, etc. Such an important feature makes computer vision a hot spot in artificial intelligence research. Character recognition is an important part of computer vision. This study aims to build an accurate handwritten character classification model. Inspired by three landmark algorithms (AlexNet, VGGNet and ResNet) in the ImageNet competition in recent years, we modified the architecture of these three original models and designed nine neural

network models for the classification task of 62 character classes recognition on the by class sub-dataset in EMNIST. The nine designed models are named AlexNet_base_1, AlexNet_base_2, AlexNet_base_3, VGGNet_base_1, VGGNet_base_2, VGGNet_base_3, ResNet_base_1, ResNet_base_2 and ResNet_base_3, of which the best performance is the AlexNet_base_1 model that has undergone data preprocessing and hyperparameter adjustment, and the accuracy on test set is 87.65%.

2 Previous work

In this section, we analyse several research papers on the EMNIST dataset and find from these efforts that neural network-based character recognition algorithms outperform some traditional algorithms in terms of recognition accuracy or recognition speed. Five papers related to the by class dataset are described below and compared with our study.

The authors who originally proposed EMNIST applied the online pseudo-inverse update method (based on OPIUM) and a linear classifier to the classification task for each subset of this dataset [7]. For the classification task of the By Class dataset, the OPIUM-based classification network iterates ten times using 10,000 hidden layer neurons to calculate the exact pseudo-inverse solution of the output weights and achieves a peak accuracy of 77.65%. The linear classifier technically differs from it in that it has no hidden layers and has an accuracy of 51.8%. In distinction to the purpose of the experiments we conducted, Cohen et al. proposed that the classification results provided in this work are intended to form a good benchmark for the dataset provided and use these results to characterize and validate the dataset. In contrast, we focused mainly on CNN-related frontier techniques in our experiments to explore high accuracy classification models.

Deep learning techniques based on neural network achieved outstanding results when tested at EMNIST. Yao Peng and Hujun Yin investigated the Markov random field-based convolutional neural network (MRF-CNN) algorithm for image classification [11]. This method extracts various features from the original input by modelling different MRFs and deriving various filters, and then trains a deep convolutional network using a prefix filter set model to perform image classification. The test by class sub-dataset yielded the 87.48% to 88.06%. The advantage of this algorithm is that the spatial smoothness of MRF affects the interaction between close random variables in space, allowing robust extraction of information from the features [12].

In an interesting study combining handwriting recognition technology with Android, Mor et al. define a common six-layer convolutional neural network model, consisting of a reshaping layer, two convolutional layers, a Max-pooling layer, a flattening layer and a dense layer. The accuracy of this model in testing the EMNIST by class dataset is 87.1% [13]. Convolutional layers are applied to the input image using a convolutional filter for feature extraction, and the method uses two convolutional layers to ensure high accuracy. Unlike the convolutional neural model, we used in the follow-up, the training time for the model in this study was 20 hours with 512 units in the dense layer, halving the time with the GPU. The authors also report using several optimizers from the keras library to train the CNN model, with accuracy peaking when using Adamax.

EvoDCNN is an evolutionary deep convolutional network cited by Tahereh et al [14]. It uses Genetic Algorithm (GA) to generate a variable-length network with high classification accuracy by setting a fixed-degree encoding model, which greatly reduces the amount of calculation and improves the training speed. It achieves an accuracy of 88.34% when tested on the EMNIST By class dataset.

Aboozar et al. used transfer learning techniques to process large imbalanced datasets and designed the AdaBoost-CNN model by combining two basic estimators, adaptive boosting capability and neural network [15]. This method based on SAMME.R [16] of updating the training with weighted probability estimates obtained a test accuracy of 87.74% on the EMNIST classification dataset.

The results of the above related studies are plotted in Table 1.

Technique	EMNIST By_Class Accuracy
Linear classifier	51.80%
OPIUM	69.71%
Markov random field CNN	87.77%
CNN(2 Conv + 1 Dense)	87.71%
AdaBoost-CNN	87.74%
EvoDCNN	88.34%

Table 1: Related work on EMNIST By_Classs

3 Methodology

3.1 AlexNet

3.1.1 Description of algorithm

The AlexNet algorithm, developed by Hinton and his student Alex Krizhevsky, won the 2012 ImageNet competition with a result of 15.3% for the proportion of test images whose correct labels were not among the five labels the model considered most likely (the top five test error rate) [17]. This deep convolutional neural network contains 60 million parameters and 650,000 neurons and the overall architecture consists of five convolutional layers and three dense layers, with the last dense layer using 1000-way softmax to output a distribution of 1000 classes (Fig. 1).

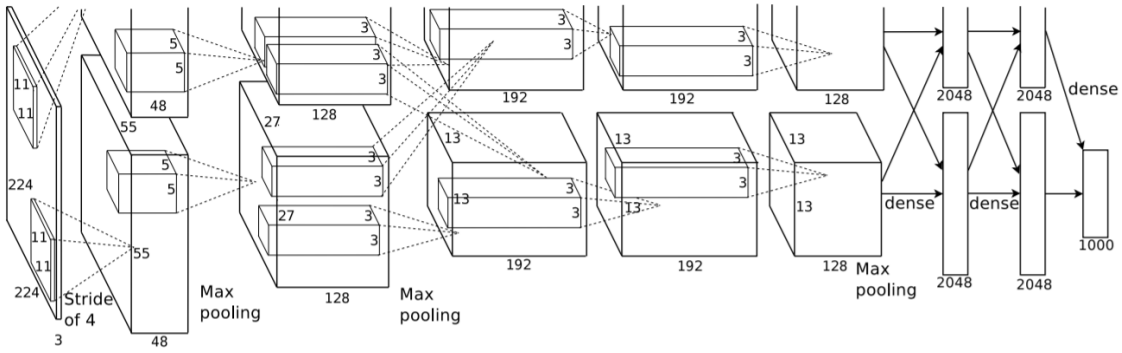


Figure 1: An illustration of the architecture of AlexNet. Citing from [17]

The six particularities of its algorithm will be described in turn in this section.

a) ReLU Activation Function

The AlexNet algorithm adopts the Rectified Linear Unit (ReLU) technique [18], a non-saturated, non-linear activation $f(x) = max(0, x)$that provides a significant improvement in training speed over neural networks using the traditional activation function $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$, and successfully solves the gradient dispersion problem of Sigmoid when trained deep network.

b) Multiple GPUs parallezation

Since the 3GB of memory in a single GTX580 GPU largely limits the maximum size of the training network, the authors distributed AlexNet across two GPUs and stored half of the cores in each GPU's video memory, using CUDA to accelerate the training of the deep convolutional network and taking advantage of the GPU's powerful parallel computing capabilities to control the performance loss of communication.

c) Local Response Normalization layer

LRN builds a competitive mechanism for the activity of local neurons, such that values with larger responses become relatively larger and suppress other neurons with smaller responses, thus enhancing the generalization ability of the model. The expression of the LRN output is

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-\frac{n}{2})}^{min(N-1,i+\frac{n}{2})} (a^j_{x,y})^2\right)^{\beta}}$$

where $a^i_{x,y}$ represents the ReLU output calculated with kernel i at position (x, y), N represents the number of total kernels in the layer, n represents the number of adjacent kernels in the same spatial location and k, n, α and β are hyperparameters, all of which are constants.

d) Overlapping Pooling

AlexNet sets the pixel spacing of the aggregate cell grid (s) to be smaller than the size of the pooling kernel (z) to obtain overlapping maximum pooling, and three maximum pooling layers are used in the model structure to avoid the blurring effect of average pooling.

e) Data augmentation

To reduce image data overfitting and improve generalization, Alex et al. used two forms of data enhancement, the first being an image transformation that randomly intercepts a $224 * 224$ sized region from the original $256 * 256$ image, which is equivalent to increasing the amount of data by a factor of 2048. The second form is PCA processing of the RGB data of the training image. The quantity $[p_1, p_2, p_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$has been add on each RGB image pixel $[I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$.

f) Dropout

The Alexnet model uses the dropout technique in first two dense layers to randomly ignore a portion of neurons by setting the probability of each hidden neuron's output is 0 to be 0.5, whilst remaining constant in the input and output layers, which can effectively prevent overfitting of neural networks.

3.1.2 Model Design

For the handwritten letter EMNIST dataset, we first proposed three improved model architectures based on the AlexNet algorithm(AlexNet_base_1, AlexNet_base_2 and AlexNet_base_3) (Table 2).

Original AlexNet	AlexNet_base_1	AlexNet_base_2	AlexNet_base_3
Conv11-96	Conv3-48	Conv5-48	Conv3-48
LRN	BN	BN	BN
Maxpool			
Conv5-256	Conv3-128	Conv5-128	Conv3-128
LRN	BN	BN	BN
Maxpool			
Conv3-384	Conv3-192	Conv3-192	Conv3-192
Conv3-384	Conv3-192	Conv3-192	Conv3-128
Conv3-256	Conv3-128	Conv3-128	
Maxpool			
FC-4096	FC-256	FC-256	FC-256
Dropout(0.5)	Dropout(0.5)	Dropout(0.5)	Dropout(0.5)
FC-4096	FC-256	FC-256	FC-256
Dropout(0.5)	Dropout(0.5)	Dropout(0.5)	Dropout(0.5)
FC-62(1000)	FC-62	FC-62	FC-62

Table 2: Model architectures based on AlexNet

Our model design is basically the same as the number of structural layers of the original Alex net model, but the size of receptive field and the number of channels in the convolutional layer are adjusted to make the neural network more suitable for training the By Class dataset of EMNIST. The main reason for choosing the AlexNet algorithm is its subversive performance. In order to speed up training, Krizhevsky et al. applied unsaturated neurons and successfully implemented efficient GPU convolution operations. This is a key moment in the history of computer vision.

The details of our modifications to the model construction and reasons will be described below.

a) Decreases of the receptive field size and the number of channels

The dataset used for training the original Alex Net network is about 1.2 million images, and the images are downsampled to a pixel size of 256*256 while our training dataset contains 814,255 images of size 28*28. Due to the huge change in image size, we halved the number of channels in the convolutional layer, and the sizes of the receptive field where the pixels on the feature map output by the first two convolutional layers are mapped on the original image are also reduced to accurately adapt to our situation.

b) Batch Normalization Applications

The LRN technology of the original AlexNet imitates biologically active neurons to suppress adjacent neurons to enhance the generalization ability of the model, but it is gradually replaced due to the small effect. For this, we adjust the parameters back to 0 to 1 with a normal distribution through the implementation of batch normalization, the connection between the parameters is unchanged to some extent, but the training speed is improved.

c) 1000-way softmax culling

Since the subset used by ILSVRC contained 1000 categories, the original algorithm used 1000-way softmax technology in the last fully connected layer to output 1000 label results, and we only involved 62 categories, hence the output in our model is 62.

3.2 VGGNet

3.2.1 Description of algorithm

VGGNet is a deep convolutional neural network proposed by the Visual Geometry Group at the University of Oxford in 2014 and won first place in the ImageNet competition that year for the localization task and second place for the classification task. The authors tried six different architectures of the VGG model with different depths, 11, 13, 16 and 19 layers[19]. The overall structure of a series of VGG models is similar, containing five sets of convolutional layers, each followed by a maximum pooling layer, and then three fully connected layers, while the differences between them are the receptive field size and the number of channels in each set of convolutional layers (Fig. 2). Each set of convolutional layers contains 2-4 convolutional operations, with a convolutional kernel of size 3*3 and a convolutional step of 1, and a pooling kernel of 2*2 and a step of 2. The most obvious improvement is the reduction in the size of the convolutional kernel and the increase in the number of convolutional layers.

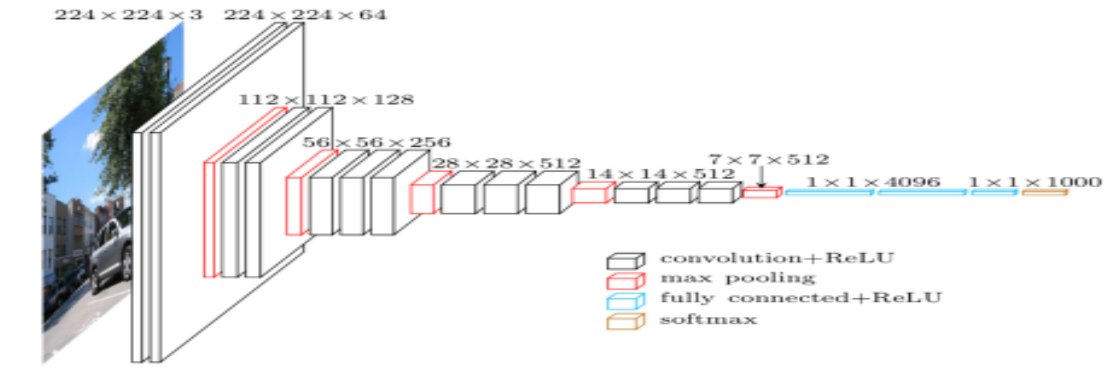


Figure 2: An illustration of the architecture of VGGNet. Citing from [19]

Features of VGGNet include:

a) Smaller convolution kernels and multiple convolution sublayers

VGG uses multiple convolutional layers with smaller convolution kernels (3x3) instead of one convolutional layer with larger convolution kernels. expressiveness of the network. VGG achieves the same performance by reducing the size of the convolution kernel (3x3) and increasing the number of convolutional sub-layers.

b) Smaller pooling kernels

Compared with AlexNet's 3x3 pooling kernel, VGG all uses 2x2 pooling kernel.

c) More channels

The number of channels in the first layer of the VGG network is 64, and each subsequent layer is doubled to a maximum of 512 channels. The increase in the number of channels allows more information to be extracted.

d) Deeper network and wider feature maps

Using continuous small convolution kernels instead of large convolution kernels, the depth of the network is deeper, and the edges are filled, and the process of convolution does not reduce the image size.

3.2.2 Model design

The VGGNet algorithm explores the effect of convolutional network depth and Table 3 reflects our thinking on the architectures of the VGG model. We propose three improved models based on VGGNet (VGGNet_base_1 , VGGNet_base_2 and VGGNet_base_3).

Original VGGNet	VGGNet_base_1	VGGNet_base_2	VGGNet_base_3
Conv3-64	Conv3-48	Conv3-48	Conv3-48
Conv3-64	Conv3-48	Conv3-48	
Maxpool			
Conv3-128	Conv3-64	Conv3-64	Conv3-64
Conv3-128	Conv3-64	Conv3-64	
Maxpool			
Conv3-256	Conv3-128	Conv3-128	Conv3-128
Conv3-256	Conv3-128	Conv3-128	
Conv3-256	Conv3-128		
Maxpool			
Conv3-512	Conv3-256	Conv3-256	Conv3-256
Conv3-512	Conv3-256	Conv3-256	
Conv3-512	Conv3-256		
Maxpool			
Conv3-512	Conv3-256	Conv3-256	Conv3-256
Conv3-512	Conv3-256	Conv3-256	
Conv3-512	Conv3-256		
Maxpool			
FC-4096	FC-256	FC-256	FC-256
FC-4096	FC-256	FC-256	FC-256
FC-62(1000)	FC-62	FC-62	FC-62

Table 3: Model architectures based on VGGNet

All three of our proposed models are based on VGG13, and we fully retain the feature of using very small receptive fields (3x3) throughout the network, but with a targeted reduction in the number of channels and convolutional layers.

The reasons we chose the VGGNet algorithm for this experiment is that VGGNet developed as a deep neural network, exceeds the baseline for many tasks and datasets beyond ImageNet. In addition, it is still one of the most popular image recognition architectures. Nevertheless, due to the size of our dataset not being suitable for using too deep a neural network model, we discarded the VGG16 model that worked best in the original report and chose VGG16 as our original model. The adjustment of the convolutional layer parameters and the number of convolutional layers within the model and the reasons are explained below.

a) Decreases of number of channels

The number of channels in the first layer of the original VGG network was 64 and was doubled in each subsequent layer to a maximum of 512 channels, which allowed more information to be extracted. The number of channels is increased so that more information can be extracted. As compared to our image size, which is much smaller than the VGGNet training data set, fewer features can be extracted, so we halved the number of channels in almost every convolutional layer of the prototype.

b) Changes of number of convolutional layers

VGGNet demonstrates that increasing the depth of the network can influence the final performance of the network to some extent. The number of EMNIST BY Class dataset and the size of the images dictated that the neural network for this dataset should not be too deep, hence we constructed 13-layer, 10-layer and 5-layer convolution layers to explore greater classification accuracy.

3.3 ResNet

3.3.1 Description of algorithm

ResNet was proposed by Kaiming et al. of Microsoft Labs in 2015[20]. They found through experiments that the deep network has a degradation problem. When the network depth increases, the network accuracy saturates or even declines. This phenomenon can be seen intuitively in Fig. 3: the 56-layer network is even worse than the 20-layer network.

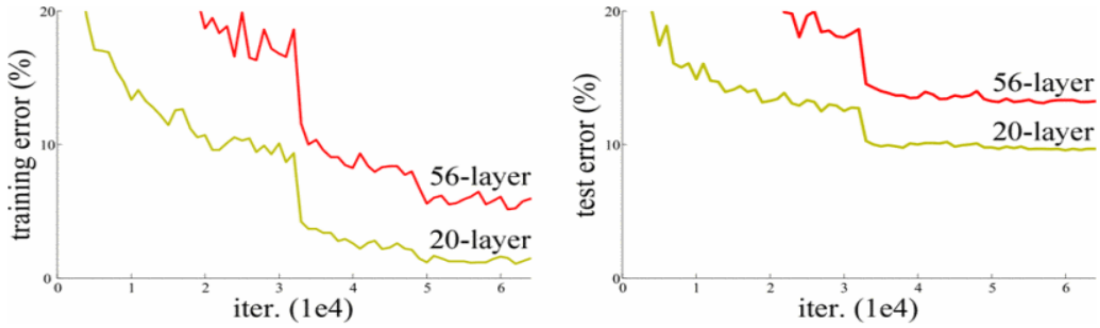


Figure 3: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. Citing from [20]

Residual learning is developed to solve the degradation problem. For a stacked layer structure, when the input is x, the learned feature is recorded as $H(x)$, and the residual $F(x) = H(x) - x$is expected to be learned, hence the learning feature is actually $F(x) + x$. The reason is that residual learning is easier than direct learning of raw features. When the residual is 0, the stacking layer only does the identity mapping at this time, at least the network performance will not be degraded, in fact the residual will not be 0, which will also make the stacking layer learn new features based on the input features, resulting in better performance.
ResNet is based on VGG19 and adds residual units through a short-circuit mechanism. The change is mainly reflected in the fact that ResNet directly uses the convolution of stride=2 for downsampling, and replaces the fully connected layer with the global average pool layer. An important design principle of ResNet is that when the feature map size is reduced by half, the number of feature maps is doubled, which maintains the complexity of the network layers. ResNet uses two types of residual units, as shown in Fig. 4. The left image corresponds to a shallow network, while the right image corresponds to a deep network. For short-circuit connections, when the input dimension same as the output dimension, they can be directly added. When the dimensions are inconsistent, there are two strategies to implement (1) Use zero-padding to increase the dimension (2) Use a new mapping (projection shortcut).

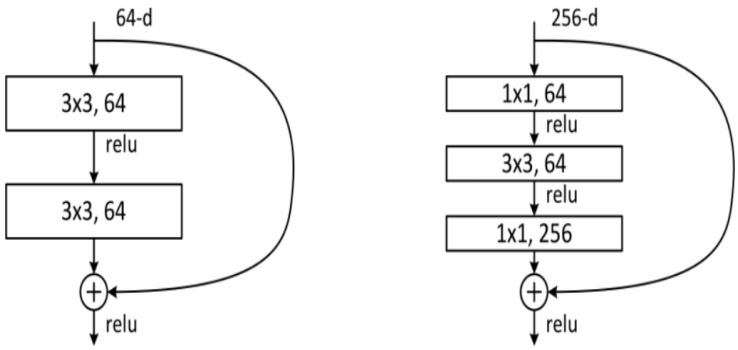


Figure 4: Left: A building block (on 56×56 feature maps) for ResNet-34. Right: A "bottleneck" building block for ResNet-50/101/152. Citing from [20]

3.3.2 Method Design

ResNet achieves excellent performance while training networks with depths of hundreds or even thousands of layers. The Table 4 below gives three architectures of convolutional neural networks that we have improved on from the ResNet network. The residual structure in the table gives the size and number of convolution kernels on the main branch, and the residual block × N indicates that the residual structure is repeated N times. We propose three improved models based on ResNet50 (ResNet_base_1 , ResNet_base_2 and ResNet_base_3).

Original ResNet50	ResNet_base_1	ResNet_base_2	ResNet_base_3
Conv7-64	Conv7-32	Conv7-32	Conv7-32
Maxpool			
$\begin{pmatrix} 1*1,64 \\ 3*3,64 \\ 1*1,256 \end{pmatrix} *3$	$\begin{pmatrix} 1*1,32 \\ 3*3,32 \\ 1*1,128 \end{pmatrix} *3$	$\begin{pmatrix} 1*1,32 \\ 3*3,32 \\ 1*1,128 \end{pmatrix} *2$	$\begin{pmatrix} 1*1,32 \\ 3*3,32 \\ 1*1,128 \end{pmatrix} *2$
$\begin{pmatrix} 1*1,128 \\ 3*3,128 \\ 1*1,512 \end{pmatrix} *4$	$\begin{pmatrix} 1*1,64 \\ 3*3,64 \\ 1*1,256 \end{pmatrix} *4$	$\begin{pmatrix} 1*1,64 \\ 3*3,64 \\ 1*1,256 \end{pmatrix} *2$	$\begin{pmatrix} 1*1,64 \\ 3*3,64 \\ 1*1,256 \end{pmatrix} *3$
$\begin{pmatrix} 1*1,256 \\ 3*3,256 \\ 1*1,1024 \end{pmatrix} *6$	$\begin{pmatrix} 1*1,128 \\ 3*3,128 \\ 1*1,512 \end{pmatrix} *6$	$\begin{pmatrix} 1*1,128 \\ 3*3,128 \\ 1*1,512 \end{pmatrix} *2$	$\begin{pmatrix} 1*1,128 \\ 3*3,128 \\ 1*1,512 \end{pmatrix} *3$
$\begin{pmatrix} 1*1,512 \\ 3*3,512 \\ 1*1,2048 \end{pmatrix} *3$	$\begin{pmatrix} 1*1,256 \\ 3*3,256 \\ 1*1,1024 \end{pmatrix} *3$	$\begin{pmatrix} 1*1,256 \\ 3*3,256 \\ 1*1,1024 \end{pmatrix} *2$	$\begin{pmatrix} 1*1,256 \\ 3*3,256 \\ 1*1,1024 \end{pmatrix} *2$
Averagepool			
FC-62(1000)	FC-62	FC-62	FC-62

Table 4: Model architectures based on ResNet50

The main reason for choosing ResNet in this experiment was to alleviate the problem of gradient disappearance associated with increasing depth in deep neural networks, reaching a new peak in the ImageNet competition of recent years. Our improvements to the original model are mainly in the number of channels, the number of convolution kernels in the residual structure and the repeated times of the residual structure.

a) Half of the number of the channels

Once again, due to the difference in datasets, the image size of 28*28 does not require 64 channels of image feature capture, which is why we have adjusted the number of channels to 32.

b) Half of the number of the kernels in each

In ResNet50, in order to reduce the amount of parameter computation, 1*1 convolution kernels are used to reduce the dimensionality of the input data and then perform the convolution operation. Since the EMNIST dataset has less data dimensionality compared to ImageNet, we reduce the number of convolution kernels for each residual block by half to ensure the integrity of the data features.

c) Different repeated times of the residual blocks

The number of repetitions of the residual block structure controls the number of channels, and the process of reducing the size of the feature maps and increasing the number of feature maps from input to output is continuously carried out. Different numbers of repetitions were tried to select the most suitable ResNet model for the classification of the EMNIST BY Class dataset, and the results of the experiment are presented in the next section.

3.4 Pre-processing approaches

To enhance the detectability of the relevant information of the images in the EMNIST dataset, the preprocessing steps of the adjustment of image background area and normalization are indispensable.

3.4.1 Image background area increased

The image data set of EMNIST is obtained after four transformation steps of the NIST data set, Gaussian Blur, ROI Extraction, Centered Frame and Resized and resampled respectively. The ROI

extraction process extracts characters that do not fill the entire image, which will cause the ends of some characters to extend to the image boundary and cannot be clearly displayed.
The process of the character subject in the image passing through the layers of filters in the neural network may be affected by the boundary filling mechanism, resulting in distortion [21]. Widening the area of blank pixels around the character subject can help avoid padding effects. We changed the image size from 28*28 to 40*40 by adding 6 pixels of blank space to each of the four sides of the image. The widened background area adjusts the proportion of characters in the image to improve the visibility of the image in the neural network.

3.4.2 Normalization

Data normalization, a routine and critical preprocessing step, was applied in this experiment. For the case of relatively concentrated data, we use min-max normalization to achieve proportional scaling of the original data. The expression shows:

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

3.4.3 Justification

The effectiveness of data preprocessing is verified by comparing the accuracy of the test set without preprocessing and with a combination of preprocessing (Table 5 and Table 7). Taking the original model as an example, the accuracy of the AlexNet model increased from 86.47% to 86.50% after using the preprocessing combination, and the accuracy of the ResNet model after preprocessing increased from 86.18% to 86.83% which proves that our preprocessing procedure is effective and indispensable. Due to the depth of the original network of VGG, it cannot be directly trained without data preprocessing when performing classification tasks on the EMNIST dataset hence it is not considered in the justification of data preprocessing, the VGGNet-based model requires data preprocessing by default.

4 Experiment

This section presents classification experiments on the EMNIST by class dataset, the comparison and evaluation of the three neural networks we designed, AlexNet, VGG, and ResNet, as well as cross-validation fine-tuning and follow-up a meaningful discussion of experimental results.

4.1 Dataset Description

Our neural network model is applied to the handwritten character dataset: By_class, a subset of EMNIST. EMNIST is an extended version of MNIST that provides a more challenging handwritten character classification task. The organization by class contains 814,255 grayscale image data of size of 28×28 pixels, consisting of 52 uppercase and lowercase letters and 10 digits in a total of 62 classes, the classification labels are arranged in the order of 0-1, a-z and A-Z, such as label 0 represents the digit 0 and 62 represents the uppercase letter Z. The EMNIST By_Class dataset comes with the training set with 697,932 observations and the test set with 116,323 observations. For the classification research perspective, the EMNIST-by class dataset is the most useful organization.

4.2 Software and hardware environment

The Jupyter notebook environment were run in Google Colaboratory Pro. Colaboratory allowed low-cost access to advanced hardware and provided Tesla P100-PCIE-16GB GPU and 25.46G RAM to entirely run code. More details of the environment are listed in appendix.

4.3 Experiments Implementation

In this subsection, the original AlexNet, VGG16 and ResNet50 neural networks are tested with multiple variants we explore through classification learning on the EMNIST subset.

4.3.1 Experiments on AlexNet

The pretrained AlexNet model was tested to compare with the modified models. The models based on the AlexNet algorithm are built by using the sequential model from the Keras [22] library and tensorflow [23] backend to recognize and classify handwritten character datasets. The implementation of the three variant models is basically the same. The input is a grayscale image with a size of 40×40 after data preprocessing, and then it goes through three blocks that composed of convolutional layers and maxpooling layers, and finally passes through three dense output layers to flatten and classify. The differences of the three improved models are described in detail in the comparison section below and the best one is selected for EMNIST character classification.

4.3.2 Experiments on VGG16

The experiment processes are similar to AlexNet, and VGGNet is also implemented by Keras of python. Three models modified according to VGG16 pre-training model. The training steps start from inputting the image after widening the pixel size of the background of grayscale image, and then filter the image features through five blocks that include convolutional layers and one maxpooling layer. The final step is to pass three dense layers to output the classification result.

4.3.3 Experiments on ResNet50

Applying the ResNet algorithm on the EMNIST dataset, we use ResNet50 as a reference and implement the three models we designed. The experimental steps first input the preprocessed image data, with a total of 62 classes, and then pass through the first stage consisting of a convolutional layer, batch normalization layer, ReLU activation function and max pooling layer, followed by four residual blocks, and finally output the classification result through the average pooling layer and a fully connected layer.

4.4 Model Comparison

4.4.1 Comparison between models of same algorithm

For the three models of our design that based on AlexNet, their structural architectures (Table 2) are all show in the previous model design section, denoted by AlexNet_base_1, AlexNet_base_2 and AlexNet_base_3 respectively. AlexNet_base_2 changes the kernel size of the convolutional layer on the basis of the design of AlexNet_base_1. The kernel size of the first convolutional layer is changed from 3×3 to 5×5 and the second convolutional layer has same modification to explore the effect of receptive field size on model performance. AlexNet_base_3 reduces a convolutional layer based on AlexNet_base_1 to verify whether the network depth has an impact on the AlexNet algorithm. The three models were trained and tested according to the above experimental steps, and the performance of each model under different preprocessing methods was compared. The comparison results (Table 5) are as follows:

Data pre-processing	None	Time(min)	Combination	Time(min)
Original AlexNet	86.47%	12.35	86.50%	21.71
AlexNet_base_1	86.48%	7.57	86.88%	10.43
AlexNet_base_2	86.62%	7.45	**86.66%**	10.53
AlexNet_base_3	86.54%	6.35	86.57%	9.56

Table 5: Accuracy comparison between the models of AlexNet algorithm with different pre-processing

The case with the best performance is bolded, which shows that the AlexNet_base_1 based on the AlexNet algorithm is more suitable for the classification of the EMNIST By_Class dataset. Therefore, the AlexNet_base_1 model will be used for fine tuning.
For the three models we designed based on VGG16, their detailed architectures are shown in Table 3, VGGNet_base_1, VGGNet_base_2 and VGGNet_base_3 represent different modified models. The Model VGGNet_base_2 reduces one convolutional layer on the third, fourth and fifth convolutional layer blocks in the architecture of VGGNet_base_1. Compared with VGGNet_base_3 and VGGNet_base_2, the operation of reducing the convolutional layer is repeated. This design is to find the most appropriate

neural network depth. After model training and testing, the performance comparison (Table 6) of each model is as follows:

Data pre-processing	None	Combination	Time(min)
Original VGG16		86.46%	46.37
VGGNet_base_1		86.64%	22.02
VGGNet_base_2		86.91%	18.05
VGGNet_base_3		**86.99%**	8.56

Table 6: Accuracy comparison between the models of VGGNet algorithm with different pre-processing

The bolded values are the best performance in the models. The better performance of VGGNet_base_3 in the VGG model indicates that the deeper network is not effective in the EMNIST dataset due to the magnitude of the observations and the size of the image. Therefore, the VGGNet_base_3 model will be used for fine tuning.
Three models implemented according to the ResNet50 model architecture (ResNet_base_1, ResNet_base_2 and ResNet_base_3). Comparing their model architectures (Table 4), we only changed the number of repetitions of residual blocks, setting each residual block to be repeated twice in ResNet_base_2. The performance comparison of the model after training is as follows (Table 7):

Data pre-processing	None	Time(min)	Combination	Time(min)
Original ResNet	86.18%	51.47	86.71%	70.05
ResNet_base_1	85.85%	28.90	86.65%	37.95
ResNet_base_2	86.49%	17.43	86.54%	22.03
ResNet_base_3	86.66%	20.43	**86.83%**	23.68

Table 7: Accuracy comparison between the models of ResNet algorithm with different pre-processing

The bolded accuracy indicates that the case with the highest accuracy is ResNet_base_3 with data pre-processing of normalization and background increase combination. Therefore, when the number of residual blocks reaches a certain threshold, the accuracy will start to decrease.Therefore, the ResNet_base_3 model will be used for fine tuning.

4.4.2 Comparison between models of different algorithms

Through the discussion of different model architectures of the same algorithm, we concluded that in the model without hyper parameter tuning, VGGNet_base_2 which changed from the prototype of VGG16 obtained the best result with an accuracy of 86.99%. The discussion of different model architectures of the same algorithm led us to compare the models between different algorithms. Comparing the best model results among these three algorithms, the accuracy on the test set is sorted from high to low as 86.99% with VGGNet_base_3, 86.83% with ResNet_base_3 and 86.68% with AlexNet_base_1. We conclude that among the models without hyperparameter tuning, VGGNet_base_2, a variation of the VGG16 prototype, achieves the best results with 86.99% accuracy.

4.4.3 Code Speeding-up

Since the training time for all initial models was no more than an hour, the idea of using multiple GPUs to achieve code parallelism did not need to be implemented. In order to speed up our code, the change of the model architecture was implemented, which is proved by the experimental results (Table 5 & Table 6 & Table 7), and the training time of the models with the highest accuracy is compared in minutes, the AlexNet-based model was reduced from 21.71 to 10.43, the VGG-based model was faster from 46.37 to 8.65, and the ResNet-based model was reduced from 70.05 to 23.68.

4.5 Hyperparameter fine-tuning

For the hyper-parameter selection, we split the original training set portion of the EMNIST By_Class dataset into training and validation sets with a ratio of 0.9 by using the hold-out cross validation

method. Three hyperparameters optimizer, learning rate and batch size are used for the best performing model in each algorithm. The effect of each hyperparameter is explained below.

Optimizer: In this experiment, we consider two optimizers, Adam and SGD. The Adam optimizer combines the advantages of the two optimization algorithms AdaGrad and RMSProp, comprehensively considers the first moment estimation and second moment estimation of the gradient and calculates the update step size.

Learning rate: The learning rate is an important hyperparameter in neural network optimization. In the gradient descent method, the value of the learning rate is very critical. The learning rate determines the speed at which the parameters reach the optimal value. If the learning rate is too large, it may skip the optimal value and cause the function to fail to converge or even diverge; on the contrary, if the learning rate is too small, low optimization efficiency may lead to slow convergence and the algorithm may fall into a local optimum.

Batch size: Batch size represents the number of samples in each batch and is often used in Batch Gradient Descent. This method requires traversing all samples in the batch at each iteration, and the samples in the batch jointly determine the optimal direction.

In case to build the most accurate classification model, we choose two common values of each hyperparameter to experiment with the optimal model for each algorithm.

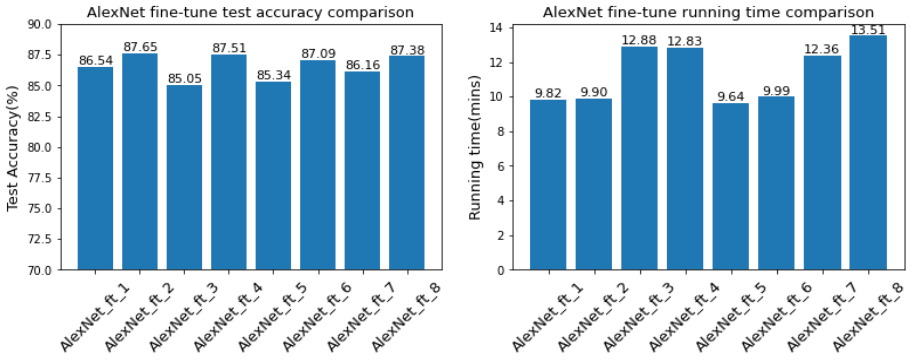
4.5.1 Fine-tuning of AlexNet_base_1

The details of the hyperparameter tuning of the model established based on the AlexNet algorithm are shown in Table 8:

Combinations of hyperparameters	Optimizer	Learning rate	Batch size
AlexNet_ft_1	Adam	0.001	128
AlexNet_ft_2	Adam	0.0001	128
AlexNet_ft_3	Adam	0.001	64
AlexNet_ft_4	Adam	0.0001	64
AlexNet_ft_5	SGD	0.001	128
AlexNet_ft_6	SGD	0.01	128
AlexNet_ft_7	SGD	0.001	64
AlexNet_ft_8	SGD	0.01	64

Table 8: Combinations of hyperparameter tuning of AlexNet_base_1

The comparison between the accuracy (Fig.5(a)) and running time (Fig.5(b)) of the test set after parameter tuning shows that the AlexNet_base_1 model has the best performance when the optimizer is Adam, the learning rate is 0.0001, and the batch size is 128 and the highest accuracy of test set is 87.65%.



(a) test accuracy (b) running time

Figure 5: AlexNet fine-tune test accuracy and running time

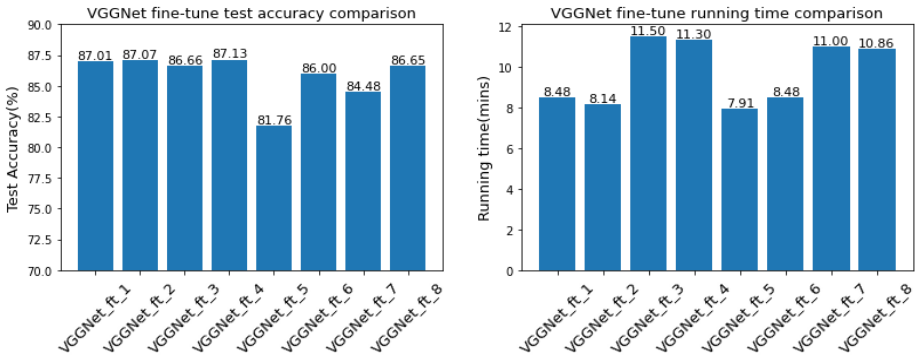
4.5.2 Fine-tuning of VGGNet_base_3

Table 9 shows the details of the hyperparameter selection.

Combinations of hyperparameters	Optimizer	Learning rate	Batch size
VGGNet_ft_1	Adam	0.001	128
VGGNet_ft_2	Adam	0.0001	128
VGGNet_ft_3	Adam	0.001	64
VGGNet_ft_4	Adam	0.0001	64
VGGNet_ft_5	SGD	0.001	128
VGGNet_ft_6	SGD	0.01	128
VGGNet_ft_7	SGD	0.001	64
VGGNet_ft_8	SGD	0.01	64

Table 9: Combinations of hyperparameter tuning of VGGNet_base_3

The following two graphs (Fig.6(a) & Fig.6(b)) make clear of the improved VGGNet_base_3 with the optimizer of Adam, the learning rate of 0.0001 and batch size of 64 and the highest accuracy of test set is 87.13%.



(a) test accuracy (b) running time

Figure 6: VGGNet fine-tune test accuracy and running time

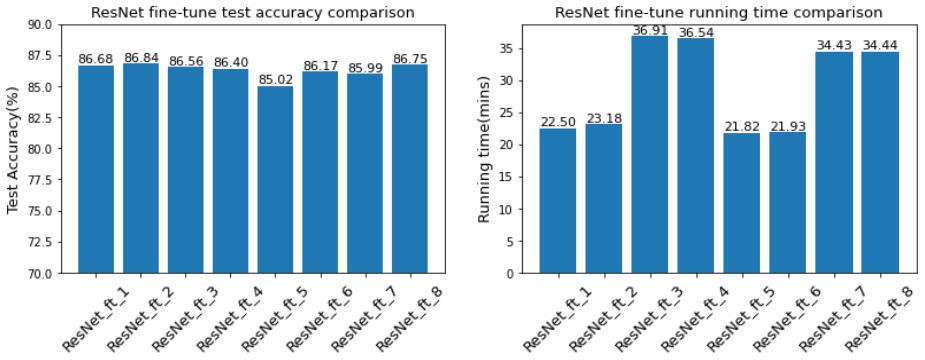
4.5.3 Fine-tuning of ResNet_base_3

The hyperparameter tuning experiments of the ResNet-based model ResNet_base_3 are defined in the following table (Table 10):

Combinations of hyperparameters	Optimizer	Learning rate	Batch size
ResNet_ft_1	Adam	0.001	128
ResNet_ft_2	Adam	0.0001	128
ResNet_ft_3	Adam	0.001	64
ResNet_ft_4	Adam	0.0001	64
ResNet_ft_5	SGD	0.001	128
ResNet_ft_6	SGD	0.01	128
ResNet_ft_7	SGD	0.001	64
ResNet_ft_8	SGD	0.01	64

Table 10: Combinations of hyperparameter tuning of ResNet_base_3

The best performance of the ResNet_base_3 model when optimizer is Adam, learning rate is 0.0001 and batch size is 128 (Fig.7(a) & Fig.7(b)).



(a) test accuracy (b) running time

Figure 7: ResNet fine-tune test accuracy and running time

The hyperparameter tuning of the three optimal models is similar, and their success is due to the fact that the Adam optimizer is more suitable for large-scale data, and the smaller the learning rate, the finer the update of the network weights. A larger batch size has a stronger generalization ability of the model, but the effect is not obvious in the VGGNet_base_3 model.

4.6 Model Evaluation

Model accuracy and running time are used as index parameters to evaluate the models. Usually, confusion matrices are used to evaluate classification tasks, but the recognition of handwritten characters has 62 classes that are not convenient to present in the report, and the confusion matrix of the best model are presented as example in Appendix.

Model	Accuracy(%)	Running time (mins)
AlexNet_base_1	87.65	9.90
VGGNet_base_3	87.13	11.30
ResNet_base_3	86.84	23.18

Table 11: Models Evaluation

Comparing the best models among the three algorithms, AlexNet_base_1 is the one with the highest accuracy and the shortest running time (Table 11). Therefore, the training model AlexNet_base_1 after fine-tuning the hyperparameters is regarded as the best model in this study, with an accuracy of 87.65%.

4.7 Discussions

4.7.1 Interpretation of experiments results and design choices

Through the above content, the experimental result is the design of AlexNet_base_1 that based on the Alex Net algorithm as the best performing model in the classification task of the EMNIST dataset, reaching an accuracy of 87.65%, followed by VGGNet_base_3 with an accuracy of 87.13%. Less ideal is the ResNet_base_3 model, which has an accuracy of 86.84%. Nevertheless, the difference between them is not significant, verifying that the convolutional neural network algorithm significantly outperforms the traditional machine learning algorithm in the classification of the EMNIST By_Class dataset.
The reason that the AlexNet_base_1 works well is probably caused by the design of two layers of batch normalization. Batch normalization replaces LRN in the original algorithm, and its advantages are
1. Speed up the convergence. Converting the data of each layer to a state where the mean is zero and the variance is 1, so that the distribution of the data in each layer is the same, the training will be easier to converge.
2. Prevent gradients from exploding and disappearing.
3. Prevent overfitting. The entire network does not learn in one direction all the time.

Although this technique is also used in Res Net models, neither the observations nor the image size of the By class dataset in EMNIST supports the training of very deep networks.

4.7.2 Strengthens and Limitations

Comparing our experimental results with the different cases discussed in the literature review applied to the EMNIST by class dataset, the accuracy of each model is shown below (Fig.8) and all three models we selected from the design achieved good results.

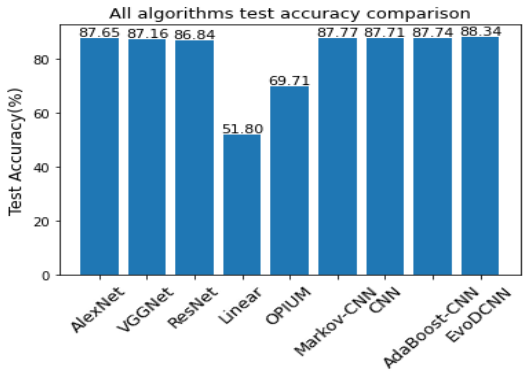


Figure 8: All algorithm comparison

By class dataset have performance defects, and the misclassifications of our test set are mostly due to the confusion of uppercase and lowercase letters, such as lowercase 'l' class, number '1' class, and uppercase 'I' class. Another subset of EMNIST, by merge, avoids this problem by merging letter categories. This direction can be tried in subsequent experiments.

4.8 Personal reflection

With assignment 2, we gain a deeper understanding of the complete process of processing image classification problems. We learned more CNN models (e.g. AlexNet, VGGNet, ResNet) that have not been carefully explained in the lecture. First, we need to pre-process the input data before training the model. An effective preprocessing technique can improve the accuracy of model classification. Second, for model selection, we need to collect similar datasets from relevant studies. From these, we can find the mainstream approaches to this kind of problem, and we can also find many novel ideas in different approaches. In model construction, we cannot directly apply the original model. Because such experiments do not improve the accuracy of the model and the study would be meaningless. Therefore, we need to improve on the models that we have collected that perform well. These changes can be made to the architecture of the model, to the internal parameters of the model, or to add new algorithms. Finally, the trained newly constructed models are compared with the previously studied models on similar datasets. The advantages and disadvantages of the improved model are analyzed. The weaknesses of the model are carefully considered to see if there are any directions for improvement and possible directions for subsequent work.

5 Conclusion

In summary, the purpose of this report is to propose an ideal handwritten character classification model that can classify a given handwritten character image of size 28×28 into 62 classifications from 0 to 9, a to z, A to Z among. These experiments are mainly implemented through data preprocessing, model architecture configuration design, experimental comparison and experimental result evaluation. The model that yields the best classification performance is AlexNet_base_1 consisting of five convolutional layers and three fully connected layers, the reasons may be the appropriate setting of the number of convolutional layers and the application of batch normalization. However, there are still potential improvements to the model:

1. On the model architecture, the convolutional layer adjustment based on the three algorithms may improve the classification performance of the model.
2. In hyperparameter tuning, a larger range can be selected so that the model can improve the classification accuracy within the allowed time consumption.
For future work, we can first consider the classes with poor classification accuracy in the experiment, such as the confusion matrix in Appendix shown in, the values of precision, recall and f1-score of classes 38, 48, 50 and 54 are all 0, the solution to this problem can help the model to effectively improve the classification accuracy. Then, algorithms with higher complexity can be considered and a classification model with higher accuracy can be trained, such as EvoDCNN mentioned in related work. For the development of character recognition technology, handwritten character recognition of scarce languages is not common in research, which is also a research direction to perpetuate culture.

References

[1] Sheikh TS, Khan A, Fahim M, Ahmad M. Synthesizing data using variational autoencoders for handling class imbalanced deep learning. In: International Conference on Analysis of Images, Social Networks and Texts. Springer; 2019. p. 270-81.

[2] Agrawal P, Girshick R, Malik J. Analyzing the performance of multilayer neural networks for object recognition. In: European conference on computer vision. Springer; 2014. p. 329-44.

[3] Ba J, Mnih V, Kavukcuoglu K. Multiple object recognition with visual attention. arXiv preprint arXiv:14127755. 2014.

[4] Ravishankar H, Sudhakar P, Venkataramani R, Thiruvenkadam S, Annangi P, Babu N, et al. Understanding the mechanisms of deep transfer learning for medical images. In: Deep learning and data labeling for medical applications. Springer; 2016. p. 188-96.

[5] Schlegl T, Waldstein SM, Vogl WD, Schmidt-Erfurth U, Langs G. Predicting semantic descriptions from medical images with convolutional neural networks. In: International Conference on Information Processing in Medical Imaging. Springer; 2015. p. 437-48.

[6] Dos Santos MM, da Silva Filho AG, dos Santos WP. Deep convolutional extreme learning machines: Filters combination and error model validation. Neurocomputing. 2019;329:359-69.

[7] Cohen G, Afshar S, Tapson J, Van Schaik A. EMNIST: Extending MNIST to handwritten letters. In: 2017 international joint conference on neural networks (IJCNN). IEEE; 2017. p. 2921-6.

[8] Vaila R, Chiasson J, Saxena V. A deep unsupervised feature learning spiking neural network with binarized classification layers for the EMNIST classification. IEEE Transactions on Emerging Topics in Computational Intelligence. 2020.

[9] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278-324.

[10] Wang W, Shen J. Deep visual attention prediction. IEEE Transactions on Image Processing. 2017;27(5):2368-78.

[11] Peng Y, Yin H. Markov random field based convolutional neural networks for image classification. In: International conference on intelligent data engineering and automated learning. Springer; 2017. p. 387-96.

[12] Dass SC. Markov random field models for directional field and singularity extraction in fingerprint images. IEEE Transactions on Image Processing. 2004;13(10):1358-67.

[13] Mor SS, Solanki S, Gupta S, Dhingra S, Jain M, Saxena R. Handwritten text recognition: with deep learning and android. International Journal of Engineering and Advanced Technology. 2019;8(3S):819-25.

[14] Hassanzadeh T, Essam D, Sarker R. EvoDCNN: An evolutionary deep convolutional neural network for image classification. Neurocomputing. 2022;488:271-83.

[15] Taherkhani A, Cosma G, McGinnity TM. AdaBoost-CNN: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. Neurocomputing. 2020;404:351-66.

[16] Hastie T, Rosset S, Zhu J, Zou H. Multi-class adaboost. Statistics and its Interface. 2009;2(3):349-60.

[17] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012;25.

[18] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. In: Icml; 2010. .

[19] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556. 2014.

[20] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770-8.

[21] Abosamra G, Oqaibi H. An optimized deep residual network with a depth concatenated block for handwritten characters classification. Comput Mater Contin. 2021;68(1):1-28.

[22] Ketkar N. Introduction to keras. In: Deep learning with Python. Springer; 2017. p. 97-111.

[23] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:160304467. 2016.

A Appendix

How to run the code

For other_algorithms.ipynb running code, our code is divided into four parts: import data, data preprocessing, model architecture and model fine-tining.

We need to run the import data section first, then run normalisation and increasing background area in data preprocessing, and then train the model. We need to run train data and validation data under model architecture to divide our data set into train and validation data. After that, there are three types of our models, namely alexnet, vggnet and resnet. When you need to verify the parameters of a certain model, you only need to run the corresponding model in model architecture, and then run the corresponding model in model fine-tining.

For best_algorithm1.ipynb, if you want to use the pre-trained model, you only need to upload the pre-trained model, then run the import data part and the data preprocessing part(Normalisation and Increasing the Background Area) in this file, and then find the last part (use the pre-trained model). Otherwise, you can run all the code retraining models in sequence.

AlexNet F1

	precision	recall	f1-score	support
0	0.68	0.87	0.76	5778
1	0.69	0.91	0.78	6330
2	0.97	0.98	0.97	5869
3	1.00	0.99	1.00	5969
4	0.97	0.99	0.98	5619
5	0.98	0.92	0.95	5190
6	0.98	0.99	0.98	5705
7	0.99	1.00	0.99	6139
8	0.98	0.99	0.99	5633
9	0.94	0.98	0.96	5686
10	0.95	0.99	0.97	1062
11	0.95	0.97	0.96	648
12	0.78	0.96	0.86	1739
13	0.90	0.94	0.92	779
14	0.96	0.99	0.98	851
15	0.78	0.97	0.87	1440
16	0.95	0.89	0.92	447
17	0.91	0.98	0.95	521
18	0.65	0.49	0.56	2048
19	0.84	0.91	0.87	626
20	0.67	0.80	0.73	382
21	0.88	0.97	0.92	810
22	0.75	1.00	0.86	1485
23	0.94	0.99	0.97	1351
24	0.69	0.49	0.58	4156
25	0.83	0.98	0.90	1397
26	0.91	0.95	0.93	413
27	0.95	0.98	0.97	809
28	0.80	0.98	0.88	3508
29	0.93	0.94	0.94	1576
30	0.77	0.98	0.86	2002
31	0.66	0.73	0.70	796
32	0.81	0.87	0.84	806
33	0.70	0.81	0.75	432
34	0.82	0.75	0.78	798
35	0.64	0.81	0.71	464
36	0.94	0.95	0.95	1644
37	0.94	0.88	0.91	853
38	0.00	0.00	0.00	432
39	0.99	0.99	0.99	1683
40	0.98	0.99	0.98	4092
41	0.59	0.06	0.10	400
42	0.81	0.61	0.70	589
43	0.97	0.95	0.96	1479
44	0.83	0.36	0.50	427
45	0.80	0.58	0.68	317
46	0.78	0.65	0.71	466
47	0.56	0.24	0.33	2535
48	0.00	0.00	0.00	464
49	0.97	0.93	0.95	1898
50	0.00	0.00	0.00	466
51	0.80	0.27	0.40	368
52	0.74	0.48	0.58	505
53	0.98	0.97	0.97	2320
54	0.00	0.00	0.00	437
55	0.96	0.95	0.96	2965
56	0.56	0.01	0.02	482
57	0.55	0.38	0.45	468
58	0.75	0.64	0.69	467
59	0.78	0.68	0.72	470
60	0.64	0.51	0.57	381
61	0.76	0.45	0.56	451
accuracy			0.88	116323
macro avg	0.78	0.75	0.75	116323
weighted avg	0.86	0.88	0.86	116323

VGGNet F1

	precision	recall	f1-score	support
0	0.68	0.87	0.76	5778
1	0.69	0.91	0.78	6330
2	0.97	0.98	0.97	5869
3	1.00	0.99	1.00	5969
4	0.97	0.99	0.98	5619
5	0.98	0.92	0.95	5190
6	0.98	0.99	0.98	5705
7	0.99	1.00	0.99	6139
8	0.98	0.99	0.99	5633
9	0.94	0.98	0.96	5686
10	0.95	0.99	0.97	1062
11	0.95	0.97	0.96	648
12	0.78	0.96	0.86	1739
13	0.90	0.94	0.92	779
14	0.96	0.99	0.98	851
15	0.78	0.97	0.87	1440
16	0.95	0.89	0.92	447
17	0.91	0.98	0.95	521
18	0.65	0.49	0.56	2048
19	0.84	0.91	0.87	626
20	0.67	0.80	0.73	382
21	0.88	0.97	0.92	810
22	0.75	1.00	0.86	1485
23	0.94	0.99	0.97	1351
24	0.69	0.49	0.58	4156
25	0.83	0.98	0.90	1397
26	0.91	0.95	0.93	413
27	0.95	0.98	0.97	809
28	0.80	0.98	0.88	3508
29	0.93	0.94	0.94	1576
30	0.77	0.98	0.86	2002
31	0.66	0.73	0.70	796
32	0.81	0.87	0.84	806
33	0.70	0.81	0.75	432
34	0.82	0.75	0.78	798
35	0.64	0.81	0.71	464
36	0.94	0.95	0.95	1644
37	0.94	0.88	0.91	853
38	0.00	0.00	0.00	432
39	0.99	0.99	0.99	1683
40	0.98	0.99	0.98	4092
41	0.59	0.06	0.10	400
42	0.81	0.61	0.70	589
43	0.97	0.95	0.96	1479
44	0.83	0.36	0.50	427
45	0.80	0.58	0.68	317
46	0.78	0.65	0.71	466
47	0.56	0.24	0.33	2535
48	0.00	0.00	0.00	464
49	0.97	0.93	0.95	1898
50	0.00	0.00	0.00	466
51	0.80	0.27	0.40	368
52	0.74	0.48	0.58	505
53	0.98	0.97	0.97	2320
54	0.00	0.00	0.00	437
55	0.96	0.95	0.96	2965
56	0.56	0.01	0.02	482
57	0.55	0.38	0.45	468
58	0.75	0.64	0.69	467
59	0.78	0.68	0.72	470
60	0.64	0.51	0.57	381
61	0.76	0.45	0.56	451
accuracy			0.88	116323
macro avg	0.78	0.75	0.75	116323
weighted avg	0.86	0.88	0.86	116323

ResNet F1

	precision	recall	f1-score	support
0	0.69	0.83	0.75	5778
1	0.69	0.90	0.78	6330
2	0.95	0.98	0.97	5869
3	0.99	0.99	0.99	5969
4	0.96	0.98	0.97	5619
5	0.97	0.92	0.95	5190
6	0.98	0.98	0.98	5705
7	0.99	0.99	0.99	6139
8	0.98	0.98	0.98	5633
9	0.94	0.97	0.95	5686
10	0.93	0.98	0.96	1062
11	0.92	0.97	0.94	648
12	0.79	0.93	0.85	1739
13	0.92	0.88	0.90	779
14	0.95	0.97	0.96	851
15	0.81	0.85	0.83	1440
16	0.92	0.88	0.90	447
17	0.88	0.96	0.92	521
18	0.64	0.50	0.56	2048
19	0.93	0.79	0.85	626
20	0.64	0.81	0.72	382
21	0.89	0.93	0.91	810
22	0.76	0.98	0.86	1485
23	0.95	0.97	0.96	1351
24	0.66	0.55	0.60	4156
25	0.83	0.95	0.89	1397
26	0.83	0.96	0.89	413
27	0.93	0.98	0.96	809
28	0.81	0.96	0.88	3508
29	0.93	0.94	0.94	1576
30	0.78	0.94	0.85	2002
31	0.68	0.73	0.70	796
32	0.76	0.92	0.83	806
33	0.64	0.88	0.74	432
34	0.80	0.71	0.75	798
35	0.65	0.77	0.70	464
36	0.93	0.93	0.93	1644
37	0.90	0.89	0.89	853
38	0.45	0.08	0.14	432
39	0.97	0.98	0.98	1683
40	0.98	0.98	0.98	4092
41	0.42	0.35	0.38	400
42	0.71	0.60	0.65	589
43	0.95	0.96	0.96	1479
44	0.77	0.39	0.52	427
45	0.77	0.73	0.75	317
46	0.78	0.61	0.69	466
47	0.53	0.24	0.34	2535
48	0.64	0.05	0.10	464
49	0.97	0.93	0.95	1898
50	0.44	0.01	0.02	466
51	0.68	0.31	0.42	368
52	0.52	0.49	0.50	505
53	0.98	0.95	0.97	2320
54	0.51	0.06	0.11	437
55	0.96	0.94	0.95	2965
56	0.44	0.09	0.15	482
57	0.56	0.44	0.49	468
58	0.81	0.49	0.61	467
59	0.81	0.54	0.65	470
60	0.56	0.54	0.55	381
61	0.77	0.39	0.52	451
accuracy			0.87	116323
macro avg	0.79	0.75	0.75	116323
weighted avg	0.86	0.87	0.86	116323

Software and hardware environment

```
[ ] ! /opt/bin/nvidia-smi
```

```
Sat May 21 10:13:33 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   36C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
[ ] !cat /proc/cpuinfo
```

```
apicid           : 2
initial apicid   : 2
fpu              : yes
fpu_exception    : yes
cpuid level      : 13
wp               : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 s
bugs             : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa
bogomips         : 4399.99
clflush size     : 64
cache_alignment  : 64
address sizes    : 46 bits physical, 48 bits virtual
power management:

processor        : 2
vendor_id        : GenuineIntel
cpu family       : 6
model            : 79
model name       : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping         : 0
microcode        : 0x1
cpu MHz          : 2199.998
cache size       : 56320 KB
physical id      : 0
siblings         : 4
core id          : 0
cpu cores        : 2
apicid           : 1
initial apicid   : 1
fpu              : yes
fpu_exception    : yes
cpuid level      : 13
wp               : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 s
bugs             : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa
bogomips         : 4399.99
clflush size     : 64
cache_alignment  : 64
address sizes    : 46 bits physical, 48 bits virtual
power management:

processor        : 3
vendor_id        : GenuineIntel
cpu family       : 6
model            : 79
model name       : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping         : 0
microcode        : 0x1
cpu MHz          : 2199.998
cache size       : 56320 KB
physical id      : 0
siblings         : 4
core id          : 1
cpu cores        : 2
apicid           : 3
initial apicid   : 3
fpu              : yes
fpu_exception    : yes
```

```
[ ] !df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
overlay         167G   39G  129G  24% /
tmpfs            64M     0   64M   0% /dev
shm              13G     0   13G   0% /dev/shm
/dev/root       2.0G  1.2G  812M  59% /sbin/docker-init
tmpfs            13G   36K   13G   1% /var/colab
/dev/sda1       174G   42G  132G  25% /opt/bin/.nvidia
tmpfs            13G     0   13G   0% /proc/acpi
tmpfs            13G     0   13G   0% /proc/scsi
tmpfs            13G     0   13G   0% /sys/firmware
```

```
[ ] !cat /proc/meminfo
```

```
MemTotal:       26692024 kB
MemFree:        23109044 kB
MemAvailable:   25007164 kB
Buffers:          142100 kB
Cached:          1897820 kB
SwapCached:            0 kB
Active:          1715324 kB
Inactive:        1564840 kB
Active(anon):    1106660 kB
Inactive(anon):      532 kB
Active(file):     608664 kB
Inactive(file):  1564308 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:               616 kB
Writeback:             0 kB
AnonPages:       1240360 kB
Mapped:           239368 kB
Shmem:              1208 kB
KReclaimable:      96644 kB
Slab:             167356 kB
SReclaimable:      96644 kB
SUnreclaim:        70712 kB
KernelStack:        5536 kB
PageTables:        17604 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    13346012 kB
Committed_AS:    3307200 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       46220 kB
VmallocChunk:          0 kB
Percpu:             3024 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      160576 kB
DirectMap2M:     5079040 kB
DirectMap1G:    24117248 kB
```