# English Character Recognition using Neural Network

Jingqi Duan
jduan27@wisc.edu

Ruoyi Cai
rcai8@wisc.edu

## Abstract

*Character recognition has been one of the popular research areas. For recognition of English characters, most previous works have focused on classifying usually a subset of English characters from images of handwritten characters, computer-synthesized images of characters, or natural images of characters. However, rarely did they focus on classifying both the uppercase and lowercase of all 26 English characters using different types of images on these characters altogether. Therefore, we selected a dataset containing three different types of images of both the uppercase and lowercase of all English characters. With this dataset, we aimed to build a model that could generalize well not only to the relatively large number of classes in our dataset but also to different types of images. After experimenting with different model architecture, our final model combined the structure of ResNet-50 with three fully connected layers. By applying momentum learning to help with convergence and image cropping to help with excluding irrelevant features from the image and help with more robust learning, this model is proved to be effective in capturing complex features in the images and handling noises in the background of the images. Using RGB input images achieved the highest testing accuracy of 96% in this model, and using binary input images could half the training time without compromising the accuracy too much. The model developed in this study can be applied to deal with word confusions and process images with word.*

## 1. Introduction

The recognition and classification of images is a very popular subject in computer science studies and other related fields. With the ability to process large datasets and construct complex learning algorithms using neural networks, deep learning provides a very powerful way to study image recognition and classification. Therefore, in this project, we will apply deep learning algorithms to develop a method to classify English letters using different types of images on both the uppercase and lowercase of all 26 English letters. The images are obtained from the Chars47K dataset[3]. The recognition of alphabets is a fascinating topic. This project attempts to find a classification model that could generalizing well for identifying every English character from different types of images. The model developed in this study can be applied to deal with word confusions and process images with word.

## 2. Related Work

Character recognition has been one of the popular research areas. Varied models are trained to classify digits and characters. Previous character classifications models usually focus on handwritten characters, font, natural images, and so on. Attigeri trains feedforward neural network to classify 26 English handwritten alphabets and gains 90.19% test accuracy at highest [1]. He also suggests a less complex system, neural network based offline handwritten character recognition method without feature extraction [1]. Wu, Yu, and Chen introduce the classification algorithm based on Support Vector Machine and the feature extraction algorithm based on Principal Components Analysis [9]. They apply the feature extraction and classification to character image recognition and achieve 85% accuracy [9]. Jadhav and Veeresh suggest a method for optical Character Recognition, which translates images of handwritten, typewritten, or printed text into a format understood by machines for the purpose of editing, indexing/searching, and a reduction in storage size [5].

As mentioned above, models for character classifications usually focus on handwritten character, font, or images separately but not altogether. In our model, the data are handwritten characters, computer-generated fonts, or natural images. Moreover, we not only try to classify English alphabets in different types, but also expect to distinguish between uppercase and lowercase, which sums to 52 classes in total.

## 3. Proposed Method

### 3.1. Multilayer Perceptron (MLP)

We start with MLP. MLP is a class of feedforward artificial neural network, which trains on a set of samples with inputs and labels. MLP is mainly involved with two paths:

forward path and backward path.

In forward path, except the input layer, the net inputs of each layer is a linear combinations of the outputs of previous layer. The linear combinations, which have two important parameters weights $w$ and bias $b$, can be expressed as:

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{x}^T \mathbf{w} + \mathbf{b}$$

Except the neurons in the input layer, each neuron has an activation function that maps the weighted sums to the output. We choose rectified linear unit (ReLU) in hidden layers and softmax regression in output layer. We also add batch normalization and dropout to each hidden layer.

In backward path, the cost is calculated and backpropagation is used to update the weights and bias to minimize the cost. The partial derivatives of the loss function with respect to weights and bias ($\frac{\delta L}{\delta w}, \frac{\delta L}{\delta b}$) are backpropagated. The weights and bias are optimized based on the negative of the gradient. We use cross entropy to compute loss and stochastic gradient descent (SGD) to minimize the cost.

The entire process of forward path and backward path is repeated until the cost is minimized.

### 3.2. Convolutional Neural Network (CNN)

Next we experiment with CNN, which is widely used for image recognition and image classification. We choose two common CNN architectures: ResNet and Inception.

**ResNet-50** ResNet, which an artificial neural network, solves the problem of vanishing gradients in the training of very deep neural networks. ResNet introduces a concept of skip connections. In ResNet, residual learning is conducted for every few stacked layers by a building block shown in Fig.1 [4]. The strength of ResNet is skipping connections, which is achieved by the block shown in Fig.1. In a building block, the identity shortcuts can be used when the input and output have the same dimensions [4]. According to He, Zhang, Ren, and Sun (2016), when the dimensions mismatch, "the shortcut still performs identity mapping, with extra zero entries padded for increasing dimension" or "the projection shortcut" is performed with $1 \times 1$ convolutions to match dimensions [4].

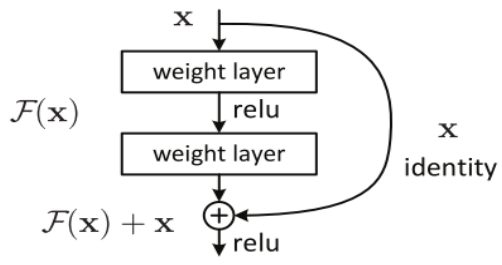


Figure 1. Residual learning: A building block [4]

We experiment with ResNet-50, the 50-layer residual neural network, as it has relatively lower operation complexity. The architecture is shown in Table 1.

| layer name | output size | ResNet-50 |
|---|---|---|
| conv1 | $32 \times 32$ | $7 \times 7$, 64, stride 2 |
| conv2 | $16 \times 16$ | $3 \times 3$ max pool, stride 2 |
| | | $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$ |
| conv3 | 8×8 | $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$ |
| conv4 | 4×4 | $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$ |
| conv5 | 2×2 | $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$ |
| | 1×1 | 1024-d fc, ReLU |
| | 1×1 | 512-d fc, ReLU |
| | 1×1 | 45-d fc, softmax |

Table 1. Architecture for ResNet-50

**Inception-v3** Inception network is an important architecture and has been improved from Inception-v1 to Inception-v4. Inception network have different sizes of filters and max pooling on the same level shown in Fig.2. Szegedy, Vanhoucke, Ioffe, Shlens, and Wojna (2016) discuss about spatial factorization into asymmetric convolutions that any $n \times n$ convolution can be replaced by a $1 \times n$ convolution followed by a $n \times 1$ convolution to decrease computational cost shown in Fig.2 [8]. Inception network also uses auxiliary classifiers to solve the vanishing gradient problem. An architecture of Inception-v3 is shown in Fig.3.

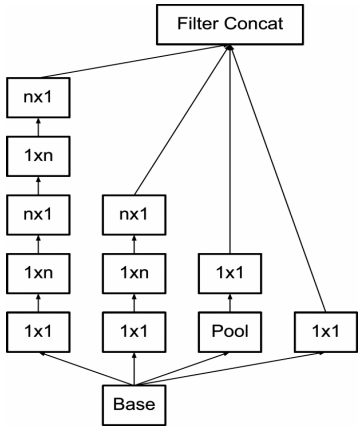


Figure 2. Inception modules after the factorization of the $n \times n$ convolutions [8]

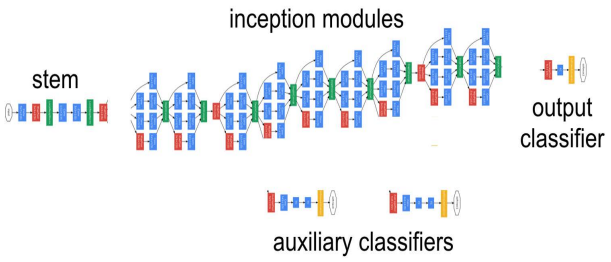


Figure 3. Inception-v3 architecture [6]

### 3.3. Minibatch Mode

In our model, stochastic gradient descent is used. Except the update of weights and bias, the entire process of minibatch mode is similar to the description in previous MLP section.

For every training epoch, weights change ($\Delta \mathbf{w}$) and bias change ($\Delta b$) are initialized to zero. Training set is divided into several batches upon the batch size. After calculating the cost and gradient, weights change and bias change are updated.

$$\Delta \mathbf{w} = \Delta \mathbf{w} + \frac{\delta L}{\delta \mathbf{w}}, \Delta \mathbf{b} = \Delta \mathbf{b} + \frac{\delta L}{\delta \mathbf{b}}$$

The ultimate weights change and bias change of a batch is subtracted from the initial weights and bias.

$$\mathbf{w} = \mathbf{w} - \Delta \mathbf{w}, \mathbf{b} = \mathbf{b} - \Delta \mathbf{b}$$

The fewer updates make minibatch mode less noisy, but the number of update is still sufficient such that convergence is not very slow.

### 3.4. Activation Functions

ReLU is used in all hidden layers. Softmax regression is used in all output layers. Softmax normalizes the activations so that they sum up to 1. Mathematically,

$$\text{ReLU}(z) = max(0, z)$$

$$\text{Softmax}(z_t) = P(y = t|z_t) = \frac{e^{z_t}}{\sum_{j=1}^{k} e^{z^j}}$$

where $t \in \{1...k\}, k =$ number of classes

### 3.5. Loss Function

Cross entropy loss function is selected. Cross entropy is commonly used for multiple class classification. Cross entropy loss increases when the predicted probability diverges from the true label. The loss is calculated as:

$$\mathbf{L} = \sum_{i=1}^{n} \sum_{j=1}^{k} -y_j^{[i]} log(a_k^{[i]})$$

where $n =$ number of features, $k =$ number of classes

### 3.6. Optimization and Regularization

**Batch Normalization (BatchNorm)** BatchNorm can increase training stability and convergence rate. It has two learnable parameters and allows the layer to learn what normalization of inputs works better for itself.

**Momentum** Momentum can make convergence faster by accelerating the gradient vecotrs in the right directions.

**Dropout** Dropout can reduce overfitting by randomly dropping units during training.

## 4. Experiments

### 4.1. Dataset

Our images are chosen from the Chars74K dataset [3]. This dataset contains more than 74K images of English characters, digits, and symbols used in Kannada script, which is a native Indian language. As recognizing Kannada characters is not practically useful under our background, we took the 62,804 images of English characters from this dataset as our dataset for this project. These images contain both uppercase and lowercase letters of the 26 English characters, and this gives 52 classes in total. The images for each letter are in three types: images of handwritten characters (Hnd), natural images of characters captured by cameras or hand-held devices (Img), and computer-synthesized images in 1016 different fonts (Fnt). This dataset would pose challenges for the learning of the neural network as it would need to classify a large number of categories and to generalize to different types of images. We randomly split the whole dataset at 6:2:2 ratio into the training dataset, validation dataset and testing dataset.

All computer-synthesized images are grayscale images in uniform shapes with $128 \times 128$ pixels, while all handwritten images are RGB images with $900 \times 1200$ pixels in each color channel. All natural images also have three color channels, but they all come in different heights and widths. Therefore, in the data preprocessing step, we unify the number of color channels, the heights and the widths of all the input images for a particular model. For comparison purpose, we trained each model with input images in three input format: RGB images, grayscale images, and grayscale images with binary pixels. To train the models using RGB images, we stack the feature map of grayscale images three times to create three color channels. To use grayscale images as model input, we convert all RGB images to grayscale. For both grayscale images and RGB images, the pixels are normalized to the [0,1] range. The other input format, grayscale images with binary pixels, are obtained from grayscale images by setting a threshold value equal to 128 to convert original pixel values into 0 if they

are below the threshold or into 1 if they pass the threshold. We decided to experiment with this input format as it seems to be common in studies involved with character recognition [2, 7], probably because a threshold function could help to filter out noises in the background of the images.

All input images for MLP models are resized to have $64 \times 64$ pixels for each color channel. In CNN models, when doing transfer learning, the input images are resized according to the requirements of the pretrained models, while in the final CNN architecture that we implemented, we decided to use images with $32 \times 32$ pixels in each color channel after taking learning speed and GPU memory limit into consideration. Finally, we randomly rotated the input images in the traning dataset to make the learning more robust.

### 4.2. MLP models

We designed two MLP models, one taking grayscale images as input, and the other one taking RGB images as input. The basic architecture of the MLP model for grayscale images takes $64 \times 64 = 4096$ inputs, converts them to 1000 features in the first hidden layers, maps them to 1000 features in the second hidden layers, and reduces these 1000 features to the number of classes in the output layer. Finally, softmax regression is applied after the output layer to obtain predicted class labels. Batch normalization is used before activations and dropout with probability of 0.2 is applied after activations in each hidden layer. The MLP model that takes RGB images as input has basically the same architectures, except that it takes $64 \times 64 \times 3 = 12288$ features as input after we simply concatenated the features in all three channels, and has 3000 and 1500 hidden units in the first and second hidden layers respectively. The model is trained using stochastic gradient descent to minimize cross-entropy. After several attempts to play around with the hyper-parameters, we found that 256 is a reasonable batch size in our case, and our models tend to perform better at relatively larger learning rate, such as 0.1. Therefore, we kept these values for our hyper-parameters and didn't bother to train our models by changing these values; instead, we played around with the optimization methods and image transformations to fine-tune our models. We trained models for 30 epochs to compare their performance, and then choose the best-performing model to train for 50 epochs.

### 4.3. CNN models

We started with transfer learning to select a common architecture that performs well on our dataset, so that we could build our model based on this architecture. Taking both accuracy and operation complexity into consideration, we decided to focus on two common architectures: ResNet-50 and Inception-v3. When implementing the transfer learning, we used models pretrained on the ImageNet, freezing the model and only changed the fully connected layer. For faster training, we simply added batch normalization, ReLU activation and dropout with probability 0.2 sequentially after the original fully connected layer in the model, and added an output layer to bring down the 1000 activations in this fully connected layer to the number of classes in our case. In this step, we only used RGB images as inputs without any image transformation and only train the models for 20 epochs for a simple comparison purpose. For ResNet-50, the input images have $224 \times 224 = 50176$ pixels in each color channel, and for the Inception-v3 model, the input images have $299 \times 299 = 89401$ pixels in each color channel.

After comparing the results from each pretrained model, we decided to proceed with ResNet-50. We implemented the same overall structure of ResNet-50, and combined this structure with three fully connected layers to further enhance the learning process. The 2048 activations from the last convolutional layer were mapped to 2048 features in the first fully connected layer maps, which were then brought down to 512 features in the second fully connected layer. Batch normalization and dropout with probability equal 0.5 were still added in the same way in the first two fully connected layers. The third fully connected layers, serving as an output layer, brought down these 512 features to the number of classes we have. Finally, softmax regression is applied on activations produced by the output layer to obtain a predicted class label. The model is trained using stochastic gradient descent to minimize cross-entropy. We used the same hyper-parameters as we did in the MLP models, and tried different optimization methods and image transformations to fine-tune our model. We still trained our model for 30 epochs to compare their performance, and then choose the best-performing model to train for 50 epochs.

### 4.4. Analyze output to improve model performance

After training the MLP models simply with stochastic gradient descent using RGB images, we found that the model accuracy is only around 58%. With grayscale images as input, the accuracy could be improved to 69%, but this accuracy is still pretty low. Therefore, we saved the predicted output from the testing dataset, and analyzed the predicted outputs from the model taking grayscale images as input to get an idea on why the accuracy is low, and how we could improve our models. By comparing the predicted output with the correct class label for each classes and searched for top predicted results for each class, we found that a major issue that affect the model accuracy is that in our dataset, the uppercases and lowercases for certain letters are hardly distinguishable, and Table 2 lists some examples of such letters. As the lowercase and uppercase of these letters have basically the same shape, it is not reasonable to train the neural network to distinguish them. Therefore, after looking for all such letters in the dataset, we decided to combine

the lowercase and uppercase of the following 7 letters into one single category: C and c, O and o, S and s, U and u, V and v, W and w, Z and z. After this correction, there are 45 classes left in our dataset.

| Class Label | Predicted Label | Frequency |
|---|---|---|
| Big C | Small c | 59.14% |
| Big S | Small s | 66.67% |
| Big Z | Small z | 20.00% |
| Small z | Big Z | 42.08% |

Table 2. Confusion between the upper/lowercases of some letters

### 4.5. Software

- We use Python to implement and train our models. The environment we used is Jupyter Notebook, Google Colaboratory, and Kaggle Notebook.
- We use R for cleaning the dataset and analyzing the output predictions of our models. The environment we used is R studio.

### 4.6. Hardware

We trained most of our models on Kaggle Notebook GPU as training on Google Colaboratory is really slow. However, we had troubles in implementing transfer learning on Kaggle Notebook, so all transfer learning is done using Google Colaboratory GPU, and this is the reason of the long training time for transfer learning shown in the result section.

## 5. Results and Discussion

### 5.1. MLP models

For MLP models with the most basic architecture, the training time and model accuracy using different input formats are displayed in Table 3. These results are obtained using the dataset before we corrected for class labels.

| Input Format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Grayscale | 1.2min | 69.242% | 69.625% | 69.41% |
| Binary | 0.76min | 71.435% | 69.694% | 69.50% |
| RGB | 1.86min | 60.203% | 58.749% | 58.73% |

Table 3. Results from the most basic MLP models using original class labels

As discussed in our experiment procedure, after observing the bad performance of the MLP models, we analyzed the predicted output and decided to combine the class labels for the uppercase and lowercase of some letters. The results

| Input Format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Grayscale | 1.15min | 81.086% | 81% | 80.4% |
| Binary | 0.5min | 81.671% | 80.44% | 80.27% |
| RGB | 1.99min | 82.485% | 82.175% | 81.44% |

Table 4. Results from the most basic MLP model using corrected class labels

obtained after training the same model but using corrected class labels are displayed in Table 4.

We could already observe a significant improvement in the model performance after correcting for class labels. This time input images in all three different formats give approximately the same model accuracy, while using binary input has noticeable advantage in training speed. Therefore, we decided to use the corrected dataset with 45 classes in all future model training. However, we still found that the training loss stopped updating at some points for our models. A plot of the minibatch costs obtained using grayscale input images is shown in Fig.4. From this plot, we can observe that the minibatch costs oscillate around 1 and could not go down further. The same issue was observed in the minibatch costs obtained using RGB input images and binary input images. Therefore, we decided to add momentum learning to help with the optimization for our cost function, as it could help with dampening oscillation and escaping local minimum or saddle points. The results obtained after we added a momentum = 0.9 into the optimization algorithm are displayed in Table 5.

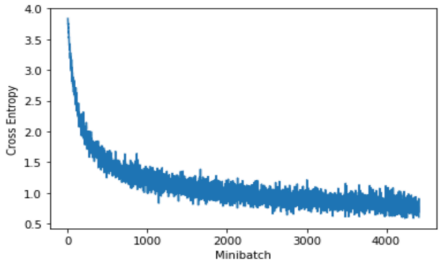


Figure 4. Minibatch costs: MLP model using grayscale input images

| Input Format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Grayscale | 0.83min | 85.383% | 85.422% | 84.72% |
| Binary | 0.57min | 85.846% | 86.445% | 85.54% |
| RGB | 1.81min | 86.628% | 86.030% | 85.63% |

Table 5. Results from MLP model after adding momentum = 0.9

After adding a momentum, the cost function dropped to slightly above 0.5, as can be observed in Fig.5, and model

accuracy is effectively improved. This proved that before adding the momentum, the optimization of our cost function probably ended up at some local minimum or some saddle points and couldn't recover. Adding a momentum helps our optimization algorithm to overcome this and thus reached a better accuracy.

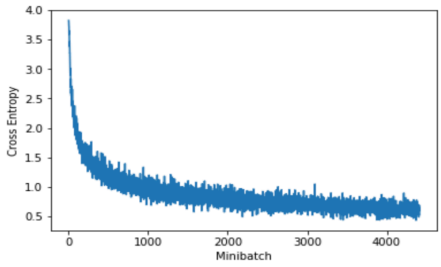


Figure 5. Minibatch costs: MLP model using grayscale input images with momentum learning

One more strategy that we used to improve our models is to crop the input images to get rid of some irrelevant features on the boarders of some natural images that might also interfere with the learning. For example, Fig.6 is a natural image for character L, but some shapes on the boarders of this image add noises to the image and lead to a wrong prediction of lowercase w. By cropping, we hope that we could remove some irrelevant information around the actual character in the images. To implement this cropping, we first resized the images to $76 \times 76$ and then crop them to $64 \times 64$. For training dataset, we used random cropping to make the learning more robust, and we did center cropping for validation images and testing images. The results obtained after we added image cropping are displayed in Table 6.

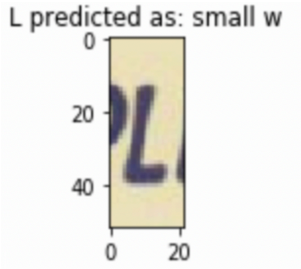


Figure 6. Wrong prediction due to noises in natural image

| Input Format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Grayscale | 1.10min | 82.148% | 87.917% | 86.84% |
| Binary | 0.52min | 82.169% | 87.577% | 87.08% |
| RGB | 2.23min | 83.310% | 89.300% | 88.20% |

Table 6. Results from MLP model after adding image cropping

Our MLP model reached the best performance so far after adding image cropping. Training models using input images with different format still lead to similar model accuracy, but using binary input images still has outstanding advantages in training speed. Therefore, we concluded that the MLP model taking cropped binary images and using momentum learning in optimization is the most ideal MLP model in our case. We trained this model for 50 epochs, and the results are displayed in Table 7. Considering the simplicity of the model structure and the high training speed, the accuracy of this model is quite satisfying. Up until this step, we have experimented with most methods to improve performance for an MLP model, and it's unlikely that we could get any more substantial improvement on prediction accuracy without changing the model structure. Therefore, in this part, we ended up with the MLP model taking binary input and employed momentum training, and we moved on to convolutional neural network.

| Input Format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Binary | 0.5min | 84.954% | 90.06% | 88.95% |

Table 7. Results after training the best-performing model for 50 epochs

## 5.2. CNN models

The results from transfer learning on ResNet-50 and Inception-v3 are displayed in Table 8 and Table 9. As mentioned in experiment procedure, for a simple testing purpose, we only use RGB images as input to test these common architectures, and only train the models for 20 epochs. We tried both of the two datasets with and without correction on class labels.

| Model | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Resnet50 | 5.96min | 97.653% | 80.478% | 80.59% |
| Inception | 8.94min | 91.653% | 77.594% | 77.85% |

Table 8. Results of transfer learning on common architectures using original dataset

| Model | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| ResNet50 | 5.87min | 98.478% | 88.557% | 88.17% |
| Inception | 6.81min | 90.081% | 84.166% | 82.89% |

Table 9. Results of transfer learning on common architectures using corrected dataset

The test accuracy of the ResNet model is higher than that of the Inception model using either the corrected dataset or the original dataset, and the ResNet model also has a

faster training process. Although the result of the ResNet model indicates potential overfitting issue, we thought that this model would still be desirable if we solve the overfitting issue. Thus, in the CNN model we implemented, we use the structure of the ResNet-50 model, and added fully connected layers after all convolutional layers to enhance learning and do classification. Also, to deal with the overfitting issues, in our own training, we still randomly rotated and randomly cropped all training images. For image cropping, we first resized images to $40 \times 40$ and then do a random crop of $32 \times 32$ for training examples and do a center crop of the same size for validation and testing examples. Finally, according to our experiences with MLP models, we still decided to use momentum learning in our optimization algorithm. We trained this model using all three types of input format and compare the training results after 30 epochs in Table 10.

| Input format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Grayscale | 1.27min | 90.933% | 92.819% | 92.67% |
| Binary | 0.76min | 89.366% | 92.755% | 92.6% |
| RGB | 1.98min | 92.19% | 95.602% | 95.03% |

Table 10. Results from the model using ResNet-50 with 3 fully connected layers

In this case, the model performance is improved significantly using ResNet-50, and the overfitting issue was resolved after we added image transformation. This proved that CNN model is more effective in feature extraction and image classification than the MLP model. Moreover, in contrast to the similar accuracy obtained from different input formats in our MLP models, the formats of input images do seem to influence the performance of our CNN model. Using RGB images as input achieve the highest model accuracy, but the disadvantage of this model is that it is very slow to train. Using binary images as input is much faster, but a trade-off is that the accuracy is a little bit lower but still acceptable. We trained the CNN model that takes RGB images and the one that takes binary images both for 50 epochs, and the results are displayed in Table 11. After training 50 epochs, the CNN model taking RGB images as input reaches a accuracy of 96%, which is fairly good. In comparison, using binary images as input for the CNN model produced a slightly lower accuracy around 93%, but has advantages in training speed.

Finally, we compared the predicted output of our best-performing model with the predicted output before we make correction to the dataset or use any optimization or regularization techniques. By doing this, we hope to get an idea on where the learning process got improved after we corrected the dataset and used a more sophisticated structure with image cropping and momentum learning. Table 12

| Input format | Time /Epoch | Training accuracy | Validation accuracy | Testing accuracy |
|---|---|---|---|---|
| Binary | 0.8min | 91.300% | 93.755% | 93.57% |
| RGB | 1.85min | 93.89% | 96.242% | 96.11% |

Table 11. Results of self-implemented ResNet-50 after training for 50 epochs

gives prediction accuracy on different types of images before we corrected the dataset using simply the MLP model taking grayscale images as input, and Table 13 gives the prediction accuracy on different types of images obtained using the final CNN model. Prediction accuracy for each type of images was improved using the CNN model, but it improve the most for the handwritten images of characters. This shows that CNN is particularly powerful in learning more subtle and complex features in the images to compensate for the large variation in people's handwriting. The prediction accuracy also improved significantly for natural images, especially when we use input images in RGB format. This shows that CNN is also effective in handling the noises in the background of the natural images. Computer-synthesized images of the characters has the best prediction accuracy in both models, probably because they are in more uniform format with minimal noises in the background, but CNN model is still more effective in learning the computer-synthesized images.

| Type | MLP Grayscale |
|---|---|
| Fnt | 74.36% |
| Hnd | 21.15% |
| Img | 52.04% |

Table 12. Prediction accuracy for each type of images using MLP model

| Type | CNN RGB | CNN Binary |
|---|---|---|
| Fnt | 97.60% | 96.52% |
| Hnd | 86.71% | 85.84% |
| Img | 88.89% | 74.89% |

Table 13. Prediction accuracy for each type of images using CNN model

We also looked at classes with low prediction accuracy using the CNN model to understand the limitations of our model. In Table14, we summarized the classes with prediction accuracy lower than 90% in our CNN model. There are still several letter with low prediction accuracy, indicating that our model still have limitations in capturing subtle differences between images, such as the difference between upper I with lower i, and probably also with lower l. This could be future research directions to further improve our

model.

| CNN RGB | | CNN Binary | |
|---|---|---|---|
| Class | Accuracy | Class | Accuracy |
| Upper I | 77.66% | Upper I | 59.45% |
| Lower p | 85.58% | Lower p | 80.77% |
| Lower l | 86.69% | Upper H | 88.45% |
| | | Upper U | 89.37% |

Table 14. Class with prediction accuracy $\leq 90\%$ using CNN model

## 6. Conclusions

In this project, we employed various architectures and techniques to build neural network models for recognizing English characters. After comparing all models, we concluded that the CNN model combining the structure of ResNet-50 and the structure of multilayer perceptron with three hidden layers is the most effective to classify images in our dataset. In contrast, MLP models are not as powerful as the CNN model, although their simpler architectures allow for faster training process. In particular, CNN model is especially effective for classifying handwritten images, where images in the same category could be very different, and for classifying handwritten images, where there are more noises in the image background. Besides, we found that in our case, using momentum learning with a relatively large learning rate helps with more effective convergence for our models, and adding image cropping is useful not only for excluding noises in images but also for more robust learning. Moreover, we found that in addition to model architecture, optimization algorithm, and image transformation, input format also has influence on the performance of our CNN model. In our CNN model, RGB inputs produce better results than grayscale inputs or binary inputs. However, in all models, using RGB images as input tend to cause slow learning process, while using grayscale images with binary pixels as input will greatly speed up the learning process without compromising the model accuracy too much. Overall, in this project, by combining the ResNet-50 architecture with three fully connected layers, using momentum learning and cropping input images, the model we built could generalize well not only to the relatively large number of classes in our dataset but also to different types of images. Future direction would be improving the model to capture more subtle difference between different characters with very similar shape, for example upper I and lower l.

## 7. Contributions

Both of us contribute to the project equally, and worked together to device ideas for building and improving our models. Specifically, Jingqi implemented MLP model, Ruoyi implemented CNN model, and Jingqi wrote the R code to analyze the prediction output of the models. For the report, Jingqi wrote introduction, related work and proposed methods. Ruoyi finished the rest of the report.

## References


[1] S. Attigeri. Neural network based handwritten character recognition systek. *International Journal of Engineering and Computer Science*, 7, Mar. 2018.

[2] T. Dash and T. Nayak. English character recognition using artificial neural network. *CoRR*, abs/1306.4621, 2013.

[3] T. Emdio de Campos, B. Rakesh Babu, and M. Varma. Character recognition in natural images. *VISAPP 2009 - Proceedings of the 4th International Conference on Computer Vision Theory and Applications*, 2:273–280, 01 2009.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] D. Jadhav and G. Veeresh. Multi-font/size character recognition. *International Journal of Advances in Engineering Technology*, May. 2012.

[6] W. Koehrsen. Facial recognition using googles convolutional neural network. 2018.

[7] D. Y. Perwej and A. Chaturvedi. Neural networks for handwritten english alphabet recognition. *International Journal of Computer Applications (0975 8887)*, Volume 20 No.7,:1–5, 04 2011.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] B. Wu, H. Yu, and X. Chen. New applications of image classification in character recognition. *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2018.