INT3404E 20 - Image Processing: Mid-term project - Full Report

Group 14

Phan Anh Duc - Nguyen Tuan Hung - Le Vu Minh

1 Introduction

For the mid-term project of the Image Processing class `2324II_INT3404E_20`, we are tasked with performing **Sino-nom character recognition (classification)** on a provided dataset of approximately 57,000 pictures of 2,130 different Sino-nom characters.

We performed analysis and pre-processing of the data with various techniques, before inputting them through various pre-trained Deep Learning models for Computer Vision tasks. After training a set of models with potent accuracies, we ensembled them with different configurations to further enhance our accuracy score.

Ultimately, our approach achieved a maximum score of **96.98%** on the provided validation set, an improvement of roughly 30% over the accuracy of the baseline Convolutional Neural Network we chose.

This report includes the following content:

- Exploratory Data Analysis
- Methods
- Experimentations and Results

2 Exploratory Data Analysis

We must first perform exploratory data analysis on the provided Sino-nom character dataset. From the discovered label distributions and characteristics of the images in the dataset, we can identify the problems we have to deal with and the possible approaches we can take to solve this Sino-nom character classification problem.

2.1 Dataset components

- The training set includes **56,813 images** of **2,130 different Sino-nom characters**.
  - It is notable that the label ID spans from 0 to 2,138, meaning that there are 9 unused IDs in that range.
- The validation set includes **1,392 images** of **595 characters**, around 28% of the possible labels.

2.2 Label distribution

Looking at the label distribution, we notice that most labels are relatively evenly distributed.

- On average, each character has 26-27 training samples.
- The median for label distribution is 28 samples per character.

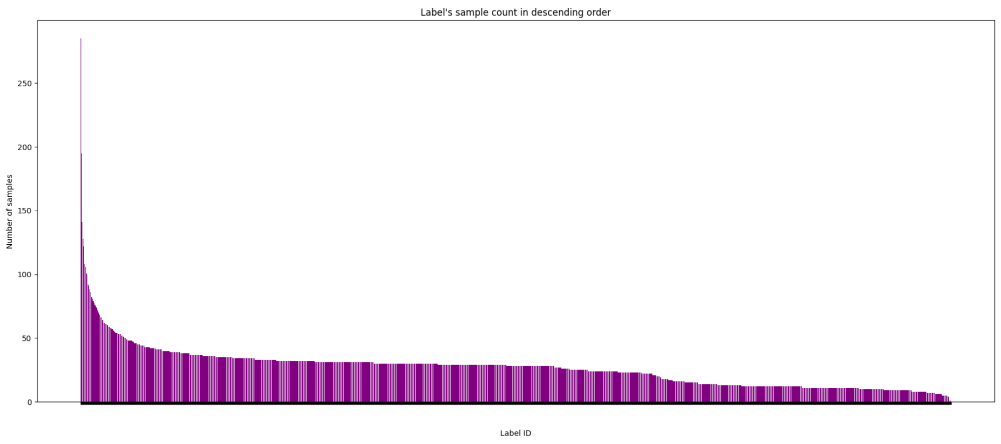

Figure 1: Label distribution in the dataset.

Towards the 2 ends of the distribution, we need to look out for labels with excessively large or low sample counts. Specifically:

- There are 226 characters with 10 or fewer samples.
- There are 30 characters with 80 or more samples.

Measures with the data or the model therefore must be taken to address the label imbalance.

2.3 Image characteristics

We proceed to sample random sets of images from the 56,813 images in the dataset to get an overview of how the images look and check for any outliers.

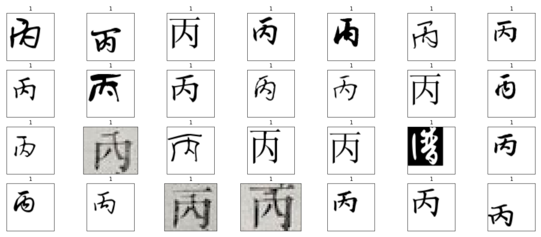

Figure 2: Black background character.

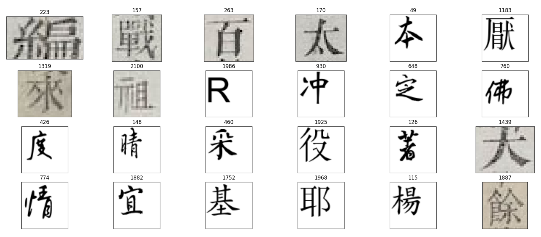

Figure 3: Roman character.

We were able to draw the following conclusions from the training set:

- There are primarily 2 types of images: "scanned" and "camera" images
  - Scanned images are pure black characters on a pure white background. The characters are clear with sharp, discernible edges. These make up around 80-90% of the training set.
  - Camera images on the other hand have a yellowish, paper background. The characters and their edges are more blurry, and with lower contrast.
- There are 2 interesting outliers that were spotted during sampling:

- Some characters are white on a black background instead. This is not fixable by just inverting the color as the black box is on top of another white background.
- During sampling, we noticed character ID 1986 is of the Roman letter "R". Checking the rest of the samples with ID 1986 shows the entire label is of this character. Therefore the model might encounter several Roman characters that it must classify as well.

Importantly, with the validation set, 100% of the data is the "camera" images. So the model must perform well on such images with lower quality and less contrast.

3 Methods

3.1 Data

3.1.1 Validation Completion

Based on exploratory data analysis, we witness a significant absence of 1535 (72%) labels compared to the training set. Therefore, to obtain an comprehensive observation of the model performance, we have to exchange part of training samples to fill in validation set.
Under dataset inspection, we reckon that some minor samples in training set and all validation set from different data sources with prefix `nlvnpf` which have different representation and features to typical image training sample such as background, strokes, image size.

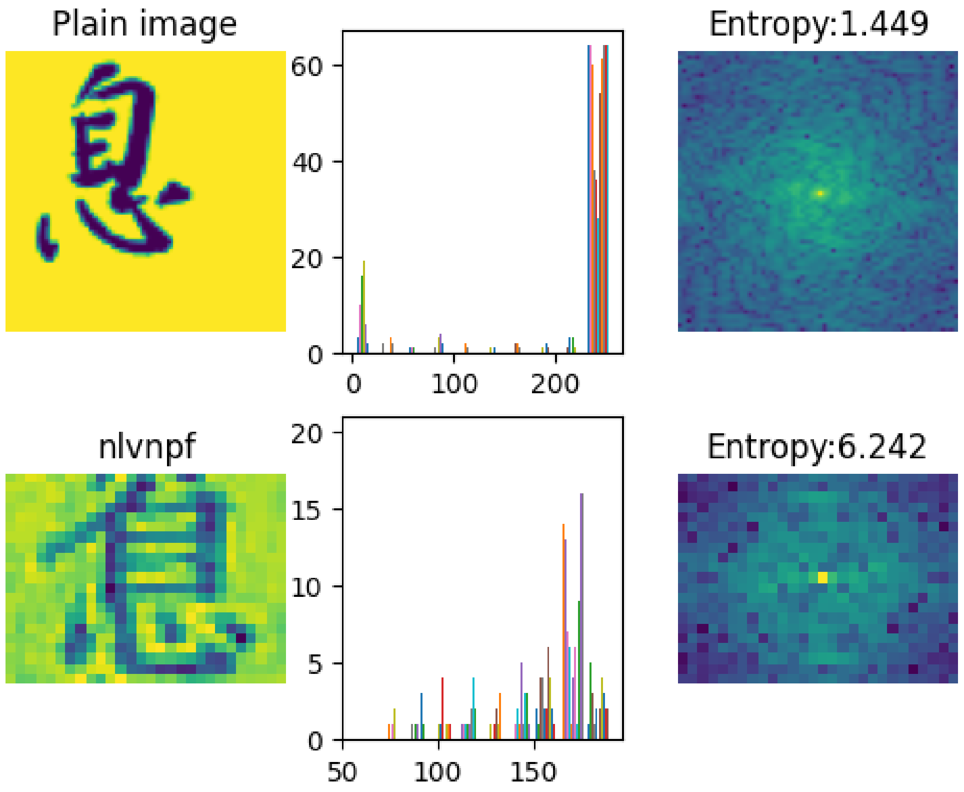

Figure 4: Difference between images (scaled) from different dataset on several benchmarks

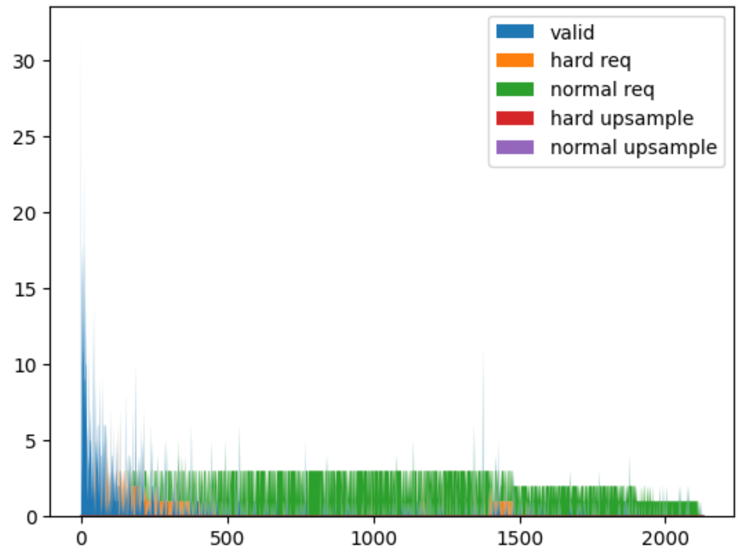

Figure 5: Available sample request on training set to complete validation set

Thus, we split the training set into typical set and the hard (nlvnpf) set, as we will query part of each set to guarantee original proportion of these set. We set the threshold for sufficient statistic empirically for each label to be the mean number of samples as 3, and tried to request from training set with maximum samples can be taken as 0.3 of the training set. For 5 scarce labels having few samples, we have to copy one of examples and perform strong non-transform augmentation to be at least observe model performance on such cases.

3.1.2 Data Augmentation

To address the difference of image features observed in 4, we perform augmentation to enhance the training set quality. Our augmentation is seperated to 2 categories applying sequentially:

- **In-place Augmentation:** We use `imgaug` library to perform color-wise, pixel-wise and piece-wise augmentations, our in-place augmentation module includes:
  - Blurring: Gaussian and Motion Blur with neighborhood of 3 pixels
  - Randomly adding or subtracting hue or saturation or value
  - Image inversion
  - Solarizing
  - Random pixel dropout
  - Jpeg Compression with strength varying between 5 and 80
- **Shape and Stroke Augmentation**: We propose a well designed augmentation module to augment on specific content features such as background, stroke or pose, this augmentation module include:
  - Content-preserving Transformation
  - Stroke Augmentation
  - Content Vanishing

3.1.3 Customized Content Augmentation and Upsampling

The proposed Shape and Stroke Augmentation pipeline includes three steps:

- **Preparation:** To make us able to conduct strong augmentation on content, we have to locate the content and its supplemental information, in which, poses and background color distribution. We use thresholding technique to extract the content mask, empirically preferring Otsu thresholding over adaptive thresholding for its robustness to noises and automatic technique. Then, we refine the content mask with morphology operations and extract line pose estimation from existing mask by morphology thinning, the result is shown in 6: On the other side, we take dilated content mask inversion as background

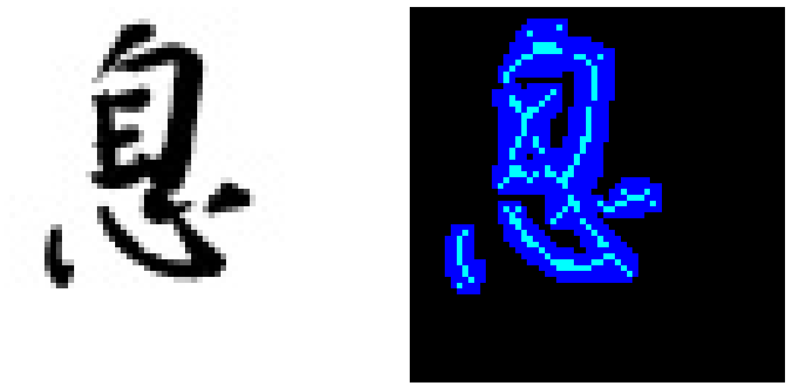

Figure 6: Extracted mask and line pose of image's content

  mask and extract background color scheme to generate a piece-wise random noise background as a cheap alternative to inpainting technique.
  As these information can be reused for each augmentation attempt, we cached it into secondary memory to optimize augmentation computational cost.

- **Content-preserving transformation**: After retrieving content mask, we compose a compact transformation with shear, rotation and scaling, with random order, using simple equation:

$$\hat{X} = (\prod_{i=1}^{n} A_i) * X = H * X$$

  For output image, we calculate image size by transforming the corners and translate them to coordinate origin, adding an arbitrary padding as translation transform factor and shrink size recovery to preserve scale transformation. Then, we apply transformation for both original image, content mask and a full mask (a image with full of 1) to further cropping out Region of Interest and performs latter augmentation. Before cropping, we generate a noise background with the same color distribution as original background, additionally, the transformed full mask with an opacity factor which ranges between (0, 1] to blend or fade the content onto the background (Content Vanishing). The Region of Interest is then cut from content mask bounding box dilated with translation factor to produce final output. By using this transformation, we can have full control on content displacement, resolution, background noise, completeness, orientation.
  Apart from augmentation, we also use it module to normalize representation of valid set by turning off all augmentation factor, the module will only crop the ROI of the image.

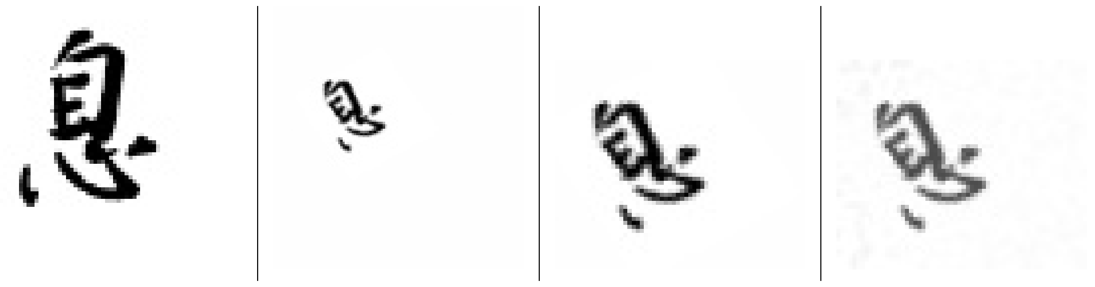

Figure 7: A brief visualization of content-preserving transformation with content 0.7 opacity and 0.7 content size

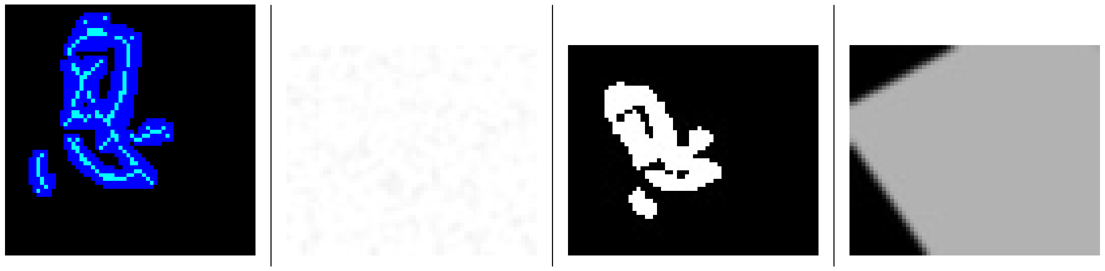

Figure 8: Visualization of underlying pipeline of content mask, background generation and full mask

- Stroke Augmentation: For better exploitation of the power of content mask, we uses the line pose of the content to either squeeze or extrude the content in the image through morphology operations.
  To thin the stroke, we dilate the line pose with cross morphology window, the size of the window is opposed to the level of augmentation.

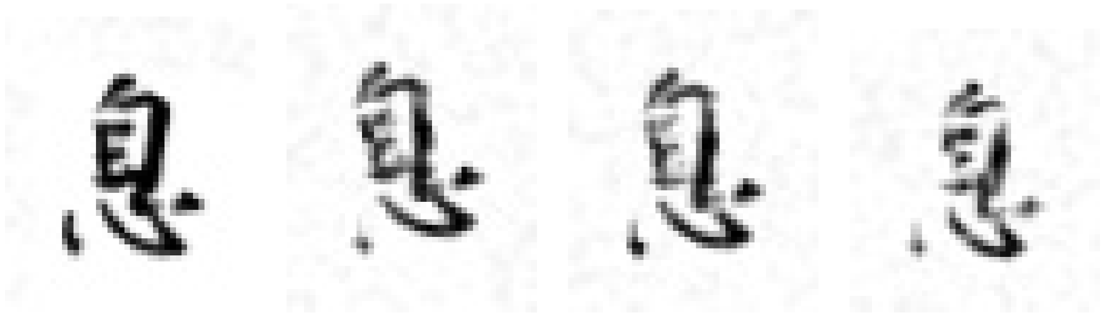

Figure 9: Visualization of content stroke thinning on many levels

On the contrary, when extruding the content, we dilate the pose with circle morphology window and multiply it with the transformed image. If inversion is intended, we inverse the region inside the extruded mask we formerly created.

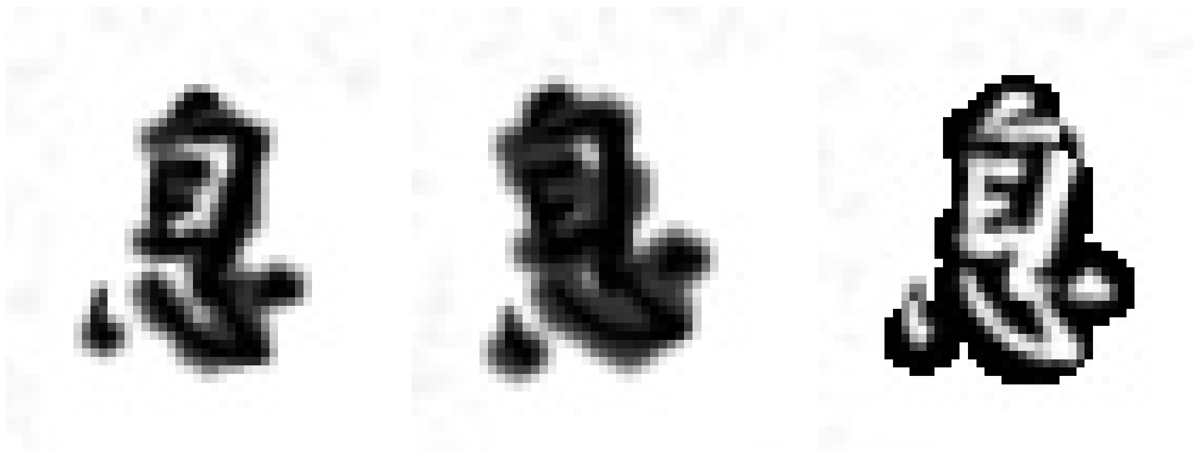

Figure 10: Stroke extrusion at two levels of augmentation and stroke inversion

Therefore, We seek a diverse pool of samples in order to enhance model's generalization capability. Besides, such augmentation abilities allow us to further up-sampling the training set with less risk of over-fitting, in which, we do up-sampling the minor labels to match the average number of samples overall. We try to implement weighted cross entropy loss but reckon that the training set and validation set share familiar distribution, which makes this optimization become redundant.

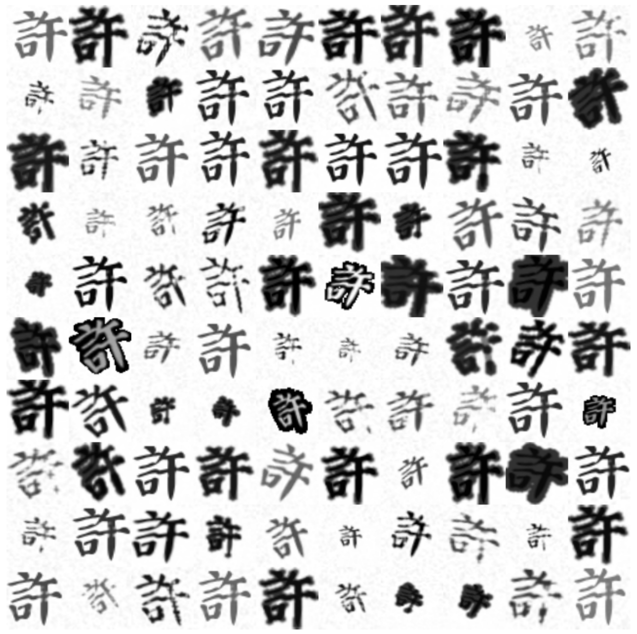

Figure 11: Example of up-sampling with custom augmentation, before piece-wise augmentation from imgaug library

3.2 Classification model

In the modern era, most image classification tasks are performed using deep learning. We relate our task to the MNIST handwritten digit classification, where studies have shown that deep learning models consistently achieve accuracy above 99%, whilst machine learning models typically suffer from higher error rates and require manual feature extraction [4].

3.2.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized type of neural network, designed to process and analyze visual data by automatically and adaptively learning spatial hierarchies of features. Inspired by the human visual system, CNNs employ a series of convolutional layers to detect patterns such as edges, textures, and shapes, making them particularly effective for tasks like image classification, object detection, etc.

CNNs extend upon traditional neural network models with the following specialized layers:

- **Convolutional Layer:** The input to a convolutional layer is a multi-dimensional array (tensor), typically representing an image or the output of a previous layer. For a grayscale image, this would be a 2D array ($h \times w$), while for a color image, it is a 3D array ($h \times w \times c$), with channels representing color depth (e.g., RGB channels). The layer applies multiple convolutional filters, often referred to as kernels, which are small matrices with learnable weights. These filters slide over the input tensor and perform element-wise multiplications followed by summations. The output is a set of feature maps, one for each filter. With N filters, the output will be a 3D array ($h \times w \times N$), with each feature map highlighting different features detected by the corresponding filter.

- **Activation Layer**: The input to an activation layer is the feature maps produced by the previous convolutional layer. The activation function, most commonly ReLU (Rectified Linear Unit), is applied element-wise. For ReLU, it replaces all negative values in the input tensor with zero, while positive values remain unchanged. The output is a tensor of the same shape as the input ($h \times w \times N$) but with the non-linear transformation applied. This introduces non-linearity, allowing the network to model more complex patterns.

- **Max Pooling Layer**: The input to a pooling layer is the feature maps from the previous activation layer. The pooling layer performs downsampling by partitioning the input into non-overlapping or overlapping regions and computing a summary statistic for each region. In max pooling, the maximum value within each region is selected. The output is a downsampled version of the input tensor. For instance, if the input is of shape ($h \times w \times N$) and you use a 2x2 pooling window with a stride of 2, the output will be of shape ($\frac{h}{2} \times \frac{w}{2} \times N$). This reduces the spatial dimensions while preserving the number of channels.

- **Fully Connected Layers**: The output of the final layer of the specialized CNN layers is flattened down to a traditional 1D tensor, which is then processed similarly to the hidden layers of a traditional neural network. At the end of these fully connected layers, a softmax activation layer yields the probability of the 2,130 possible characters the input image belongs to. The class with the highest probability is chosen as the final prediction of the model.

In our experimentations, we used the following CNN architecture, a variation of LeNet by Yann LeCun - one of the pioneers of CNN in image classification [2], which scored an accuracy of **98.64%** on the MNIST digit classification task. We selected this architecture as our baseline which we will attempt to improve with our models.

The model's architecture is as follows:

- **Convolutional Layer 1:** 3 input channels → 16 output channels, 3 × 3 kernel, stride 1, padding 1, ReLU activation
- **Max Pooling 1:** 2 × 2 kernel, stride 2
- **Convolutional Layer 2:** 16 input channels → 32 output channels, 3 × 3 kernel, stride 1, padding 1, ReLU activation
- **Max Pooling 2:** 2 × 2 kernel, stride 2
- **Fully Connected:** 32 × 16 × 16 input features → 128 output features, ReLU activation
- **Softmax Classification:** 128 input features → 2130 output probabilities, softmax activation

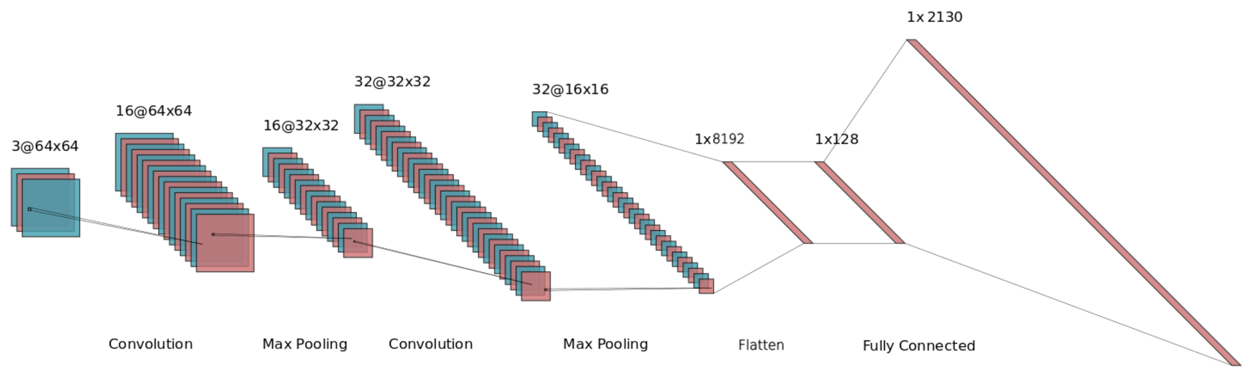

Figure 12: A visualization of our baseline CNN model.

The following models are essentially extensions of this basic Convolutional Neural Network architecture, with more layers and advanced techniques for better modeling of image features.

3.2.2 Visual Geometry Group (VGG)

Proposed in 2014, the VGG family of neural networks greatly extends upon the baseline CNN by employing up to 19 layers of weight layers [3], a big jump over the 4 weight layers in that of our baseline CNN model.

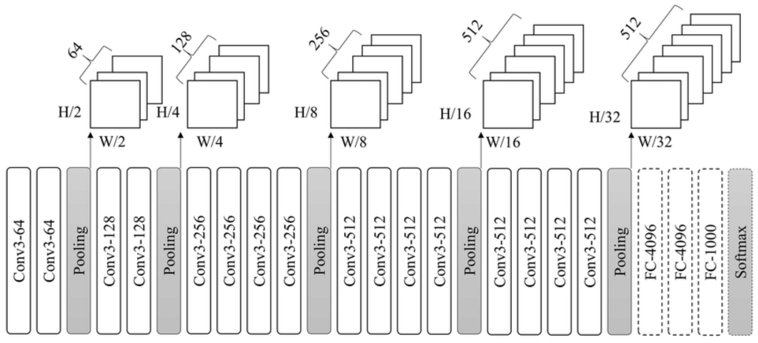

Figure 13: A visualization of a VGG-19 architecture.

The VGG family of models features 4 different configurations, VGG-11, VGG-13, VGG-16, and VGG-19, with the number of weighted layers corresponding to the number in the model name. We selected VGG-16 and VGG-19 for our experimentation for maximum feature learning capability.

However, a problem with the VGG family of models is their large number of parameters, notably in the classifier section with 2 layers of 4096 fully connected neurons. In our experiments, all models have a parameter count ranging from 30M-55M, which requires a great amount of computation to train. As a result, all models take up over 500MB in terms of weight.

3.2.3 Deep Residual Networks (ResNet)

Proposed in 2015, the ResNet family of models introduced the concept of skip connections, and bottleneck layers into the convolutional neural network, with the aim of reducing vanishing gradients and parameter counts [1].

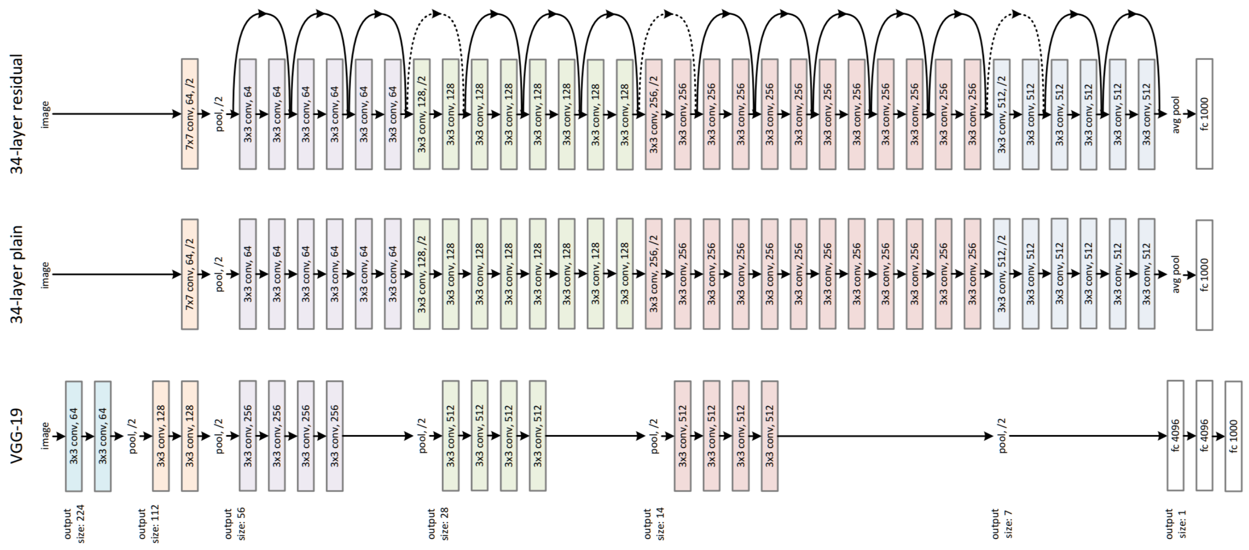

Figure 14: A visualization of the ResNet34 architecture, compared to VGG-19.

Skip connections involve adding the original input $\mathbf{x}$ with the transformed version of itself $\mathcal{F}(\mathbf{x}, \{W_i\})$ after neural network layers, essentially performing "identity mapping":

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

If the transformed input does not match the dimensions of the original input, then a projection matrix W_s is applied to the original to match the dimensions of the output:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

Bottleneck layers are 1×1 convolutional layers that can be used to reduce channels, thus reducing the number of learnable parameters in the model. This allows ResNet models to have far more weight layers, up to 152, while retaining the small number of parameters and performance equivalent to that of the VGG family.

We chose ResNet18, ResNet34, and ResNet50 for our experimentations.

3.3 Ensembling technique

In our study, we employed a soft ensemble technique to enhance classification performance. Soft ensemble methods involve combining the predictions of multiple models by averaging their outputs, specifically their logits, with assigned weights. This approach leverages the strengths of various models, aiming to produce a more robust and accurate final prediction compared to any single model alone.

4 Experimentations and Results

4.1 Standalone models

All of our models were developed in PyTorch Lightning using the Hydra template. All models, except the baseline CNN, were pre-trained on the ImageNet IMAGENET1K_V1 dataset.

Notably for ResNet18 and ResNet50, we also tried freezing all layers but the final few. Freezing parameter layers reduces their ability to learn new patterns but greatly improves training time. We observed how well the pre-trained models can perform if only given a few layers to adapt to the Sino-nom character classification problem.

We trained each model 2 times, with one run on the set of original, unaltered data, while the other run was on an online-augmented and upsampled set to address the label imbalance of the provided dataset. The training process was performed on an NVIDIA RTX 5000, with training time varying from 20 to 30 minutes, depending on the model. Our training configurations are as follows:

- **Epochs:** 40
- **Optimizer:** Adam
- **Loss function:** Cross Entropy Loss
- **Scheduler:** ReduceLROnPlateau

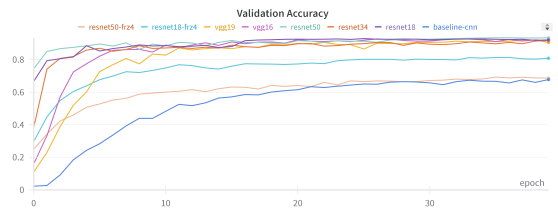

Figure 15: Validation Accuracy.

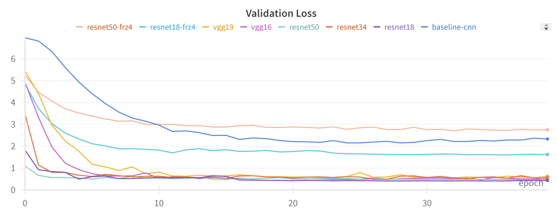

Figure 16: Validation Loss.

During the training process, we also keep track of the progression of the Loss and Accuracy on the validation set, shown in Figure 15 and Figure 16. It can be seen that our complex models converge at around Epoch 10, while the baseline CNN takes until Epoch 20 to converge.

Our training results are shown in Table 1. It is immediately apparent that the baseline CNN models and the semi-frozen ResNet models did not perform as well as the rest, with their accuracies not reaching 90% in both training settings. Notably, ResNet50 was within only 1% of the baseline CNN's performance, suggesting that inhibiting learning ability is an extremely poor trade-off for better training time.

The numbers also show that our data augmentation experiments have good effects on the performance across all models. Our 3 aforementioned "weak" models saw increases of nearly 10% in validation accuracies. Our well-performing models also benefited from this increase, with 2-4% increases in accuracy. We identified VGG-16 as our strongest model, reaching **96.16%** in validation accuracy. The prediction analysis of this model is documented in the Appendix.

	Without augmentation	**With augmentation**
Baseline CNN	0.6774	0.7716
ResNet18 (fine-tuning 4 final layers)	0.8082	0.8862
ResNet18	0.9289	0.9464
ResNet34	0.9167	0.9534
ResNet50 (fine-tuning 4 final layers)	0.6861	0.7729
ResNet50	**0.9339**	0.9511
VGG-16	0.9267	**0.9616**
VGG-19	0.9203	0.947

Table 1: Experimentation results of standalone models

4.1.1 VGG16_V2

Following training and analysis, we took our best-performing model, VGG-16, and retrained it with more advanced upsampling, and with the Weighted Cross Entropy loss function. This experiment yielded a validation accuracy of **96.15%**, which was essentially the same as the performance on the default VGG-16 training setting. We refer to this model as VGG16_V2.

4.2 Model ensembling

Model Combination	**Accuracy**
ResNet18, ResNet34, ResNet50, VGG-16, VGG-16_v2, VGG-19	0.9662
ResNet18, ResNet50, VGG-16, VGG-16_v2, VGG-19	0.9661
ResNet18, ResNet34, VGG-16, VGG-16_v2, VGG-19	0.9680
ResNet18, ResNet34, ResNet50, VGG-16, VGG-19	0.9669
ResNet18, ResNet34, VGG-16, VGG-19	**0.9698**

Table 2: Experimentation results of model ensembling

For our classification task, we utilized six different models: ResNet18, ResNet34, ResNet50, VGG16, VGG16v2, and VGG19. By calculating the weighted average of the logits from these models, we observed a notable increase in accuracy, ranging from 1-2%, as shown in Table 2. This improvement demonstrates the efficacy of the soft ensemble method in harnessing diverse model architectures to enhance overall performance.

However, further analysis revealed that the combination of ResNet18, ResNet34, VGG16, and VGG19 provided the most optimal results, scoring **96.98%** on the validation set. ResNet50 and VGG16_v2 proved to be the models that reduced the overall performance of our soft ensembling strategy.

In order to avoid over-tuning, we used equal weights for all the models in the ensemble. Overall, our findings underscore the value of soft ensemble techniques in classification problems, highlighting their potential to improve model performance by effectively integrating the strengths of various neural network architectures.

5 Task assignments

1. **Nguyen Tuan Hung - 21020205:**
    - **Group Leader**
    - **Program the Data Pipeline for the Project:** Develops and optimizes the data pipeline components, including the datamodule, collator, and sampler, to ensure efficient data processing and loading for model training.
    - **Perform Data Augmentation and Upsampling:** Implements advanced data augmentation techniques to increase the diversity of the training dataset and applies upsampling methods to balance the dataset, enhancing model performance and generalize ability.
2. **Phan Anh Duc - 21021481:**
    - **Perform basic data pipeline:** Basic data augmentation techniques, composing augmentation module from `imgaug`
    - **Implementing the Pipeline for the Project:** Works on integrating and deploying the data pipeline within the overall project framework, ensuring seamless data flow and compatibility with different components.
    - **Research and Implement Ensembling Strategies:** Investigates various ensembling techniques and selects appropriate strategies to combine the outputs of multiple models, aiming to improve prediction accuracy and robustness.
    - **Git Supervisor:** Managing github repo, commit request and merging, keeping the repo hygiene and ordered.
3. **Le Vu Minh - 21020649:**
    - **Perform Exploratory Data Analysis on the Sino-nom Dataset:** Conducts thorough exploratory data analysis (EDA) to understand the characteristics and patterns within the Sino-nom dataset, identifying key insights that inform model development.
    - **Program, Train, and Experiment with Different CNN, ResNet, and VGG Models:** Develops and trains various convolutional neural network (CNN) architectures, including ResNet and VGG models, to evaluate their performance. Conducts experiments and analysis to fine-tune model parameters and improve classification accuracy.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition.* 2015. arXiv: `1512.03385 [cs.CV]`.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[3] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2015. arXiv: `1409.1556 [cs.CV]`.

[4] I. M. Tuba, U. M. Tuba, and M. Đ. Veinović. "Classification methods for handwritten digit recognition: A survey". In: *Vojnotehnički glasnik* 71.1 (2023), pp. 113–135. DOI: `10.5937/vojtehg71-36914`.

Appendix

We looked further into the prediction results of our best standalone model, VGG-16, on the validation set and analyzed our models' performance over the challenge of label imbalance and training sample deficiency.

label	**precision**	**recall**	**f1-score**	**support**	**training_sample**
1677	1.000000	1.000000	1.000000	32	285
1985	1.000000	1.000000	1.000000	23	108
1439	1.000000	1.000000	1.000000	18	120
1748	1.000000	0.888889	0.941176	18	101
1616	1.000000	1.000000	1.000000	18	134
3	1.000000	1.000000	1.000000	15	141
1895	1.000000	1.000000	1.000000	14	92
942	1.000000	1.000000	1.000000	14	68
1118	1.000000	1.000000	1.000000	13	122
170	1.000000	1.000000	1.000000	13	71
accuracy			0.96	1392	
macro avg	0.93	0.93	0.92	1392	
weighted avg	0.96	0.96	0.9	1392	

Table 3: Top-10 labels in validation set result, generated with `scikit-learn`'s classification report.

Looking at the top 10 most frequent labels in the validation set, our models achieved perfect F1 scores in 9 out of 10 labels, with only label 1748 seeing missed 2 predictions.

label	**val_acc**	**val_freq**	**samples**
450	1.0	1	4
463	1.0	1	7
1090	1.0	2	7
1063	1.0	2	8
1955	1.0	1	9
173	1.0	1	9
1292	1.0	2	9
1202	1.0	1	9
11	1.0	1	9
1470	1.0	1	10

Table 4: Classes with the highest accuracy but with the least training samples

label	**val_acc**	**val_freq**	**samples**
1081	0.0	1	57
916	0.0	1	43
940	0.0	5	40
1548	0.0	1	39
219	0.0	2	34
1792	0.0	1	32
1104	0.0	1	31
1134	0.0	1	31
1929	0.0	1	30
636	0.0	1	30

Table 5: Classes with the lowest accuracy but with the most training samples

We also analyzed the effect of the number of training sample counts on accuracy. It can be seen in

Table 4 that even the class labels with fewer than 10 training samples, an initial concern of ours, are being predicted correctly. This confirms that our upsampling and augmentation techniques have helped resolve the label imbalance issue.

However, there are classes with over 30 samples, yet the model still fails to recognize them completely, as seen in Table 5. Notably, character label 940 appeared 5 times and were misclassified all 5 times, despite 40 training samples. Upon closer inspection, we notice that our models consistently misclassify it for character label 703. We therefore sampled and visualized the images from these 2 classes.

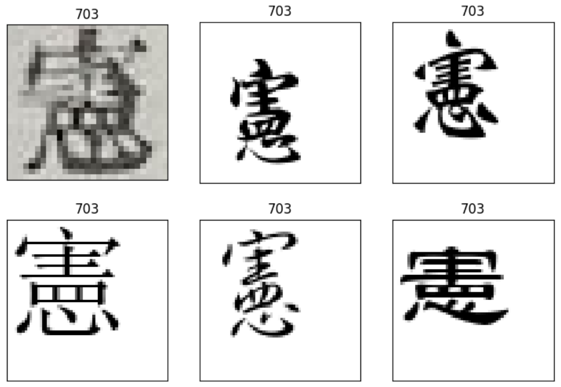

Figure 17: Samples of label 703

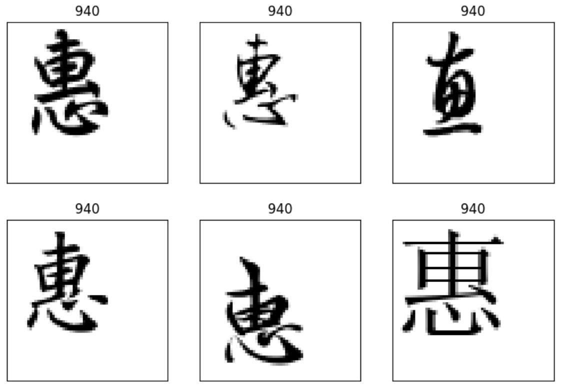

Figure 18: Samples of label 940

The visualizations in Figure 17 and Figure 18 show us a striking resemblance between the two characters, especially with the bottom pen strokes. We acknowledge the difficulty in distinguishing the characters even from a human perspective, much less a CNN feature extractor.