Classification of Mahjong Tile Image with Features Extracted by Denoising Convolutional Autoencoder

Ng Chi Him
SID 20420921
chngax@connect.ust.hk

Wong Hiu Nam
SID 20425804
hnwongab@connect.ust.hk

Abstract

Mahjong is a Chinese board game that uses a set of tiles of 42 classes. This project proposes a neural network model to predict class labels for mahjong tile images. As there exist no public dataset of mahjong images, a new dataset was collected for the project. To tackle the problem of limited number of tagged samples, we first train a denoising convolutional autoencoder that reconstruct clear images from noise-embedded input, which is shown by previous researches to be an effective unsupervised method of feature learning. The resultant feature encoder layers are then extracted and transferred to a fully-connected model for label prediction. The studied semi-supervised learning method can be applied in other image classification cases. Whereas the resultant neural network model can be deployed in helper applications for the mahjong game such as score calculation and tile counting.

1. Introduction

Mahjong is a Chinese board game, where four players plays a hand of 13 tiles and try to complete a legal hand using the 14th drawn tile. There are standard rules about how a piece is drawn, discarded and how the score of a legal hand is calculated.

The aim of this project is to build a neural network model to classify images of mahjong tiles. The model will take image of a single mahjong tile and predict a class label out of 42 possible classes.

The resultant model of this project could be used in applications for assisting players of the mahjong game. One potential application is automating the score calculation, where a claimed hand is validated and scored, which is typically difficult and time consuming due to the complicated ruleset. Another potential application is player assistance, where tiles on the table can be counted and summarized for players, as well as announcing new tiles on the table.

As to be discussed below, one major limitation of this project is that the number of available labeled mahjong tile images is small, thus semi-supervised learning is used when implementing our model. The approaches and findings of this project could be useful in other image classification problem domains where the available dataset size is small.

2. Related Work

Previous researches [1] [2] suggested that denoising autoencoders can be used to extract image features that are robust to input variances and that the method can be used as an unsupervised way to initialize layers for learning tasks like classification. Another study [3] suggested that the said approach can be improved by using convolution and max-pooling layers to build convolutional autoencoders, and reported good results on MNIST and CIFAR10 benchmarks.

While the direction of this project is based on and similar to the approaches proposed and validated by the aforementioned studies, we will be implementing models for a vastly different dataset, where image dimensions are larger and number of label classes are higher. Also, for generalizing the model and avoid overfitting our limited labelled dataset, we will not retrain the transferred encoder as it has seen a boarder range of in-domain samples.

3. Data

As there is no existing public dataset of labelled mahjong tile images available, a new dataset was collected and tagged for this project. Only part of the samples were labelled in the dataset due to time and manpower constraints. To tackle this limitation, semi-supervised learning approach was used to build the model, which will be discussed in the methods section below.

To collect the images, we leveraged online sources like Google Search and multiple online stores including eBay, Alibaba and Taobao. The raw images are then

segmented into each tiles manually using a freeware Cyotek Slicr, which is designed to split images in a grid layout. After that, each tile image is resized to 240x320 using an open-source command line tool called imgp. Samples are then tagged using an online platform Labelbox, with labels exported into the csv format.

Before training, the index of images and labels is loaded, shuffled and split into training, validation and testing sets using Pandas and Numpy. Tensorflow dataset api is then used to create a data pipeline to efficiently read and preprocess the images.

To augment the dataset and increase sample size, images were rotated by 90, 180, and 270 degrees. This could allow the model to identify rotated tiles, which is useful in real life where the player may not always place tiles right-side-up.

After augmentation, the final dataset consists of 6712 mahjong tile images, of which 2516 samples were labelled. 70% of the samples were used for training, 15% for validation and 15% for testing.

4. Methods

The proposed neural network model consists of two main parts, a feature encoder that extract features from input images and a classifier that predict class labels using encoded feature representations. The feature encoder is transferred from a denoising convolutional autoencoder, which is trained using untagged tile images. Whereas the classifier is trained using labelled samples as in conventional training methods. The motivation of employing this semi-supervised learning approach instead of conventional supervised learning is that the number of labelled image sample is limited and could result in poor accuracy and generalization if the latter approach is used.

4.1. Denoising Convolutional Autoencoder

The denoising autoencoder is built with groups of convolution and max-pooling layers to downscale the input image of size 320x320x3 into a bottleneck of size 16x16x16, which is then connected to deconvolution layers to upscale the logits back to original image size. Random gaussian noise is added to the input, where the loss function is defined to be the mean squared error between the output of the model and the original clear image. SELU, as proposed in [4], is chosen as the activation function of internal layers while sigmoid is used for the output layer. The decision of choosing SELU instead of conventional options like ReLU will be discussed in the experiments section below.

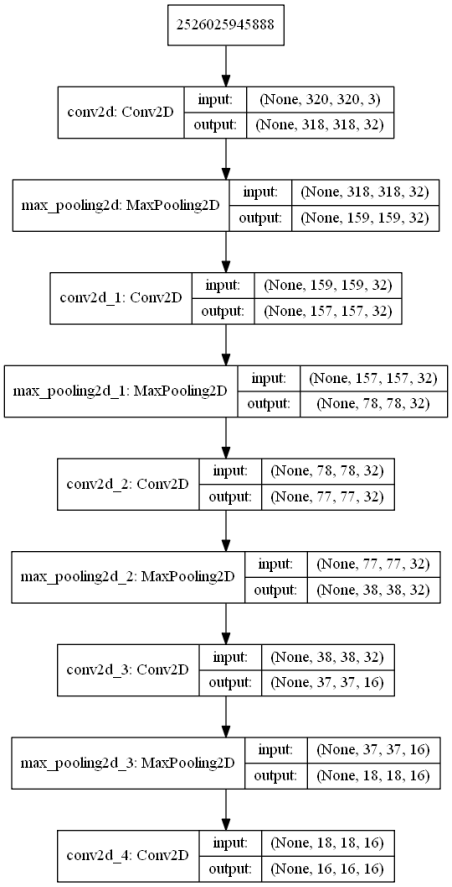

Figure 1: Encoder Design

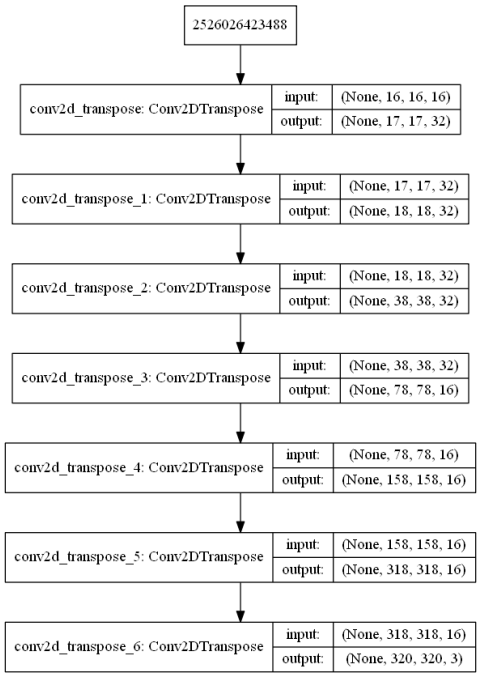

Figure 2: Decoder Design

4.2. Fully-connected Classifier

After the autoencoder is trained, the encoder portion, i.e. layers before the bottleneck, is extracted and transferred as the base of classifier model. For the classifier model, the incoming feature representations of size 16x16x16 are first processed by max pooling, batch normalization and flattening layers. The resultant 1024 logits are then passed into two fully connected layers of size 512 and finally a output layer to generate predictions. Similar to the autoencoder, SELU is used as the activation function of fully connected layers while SoftMax is used for the output layer. Dropout is added to the model for regularization. In particular, alpha dropout is used after SELU activations as suggested by the original paper [4] in order to preserve the self-normalizing effect.

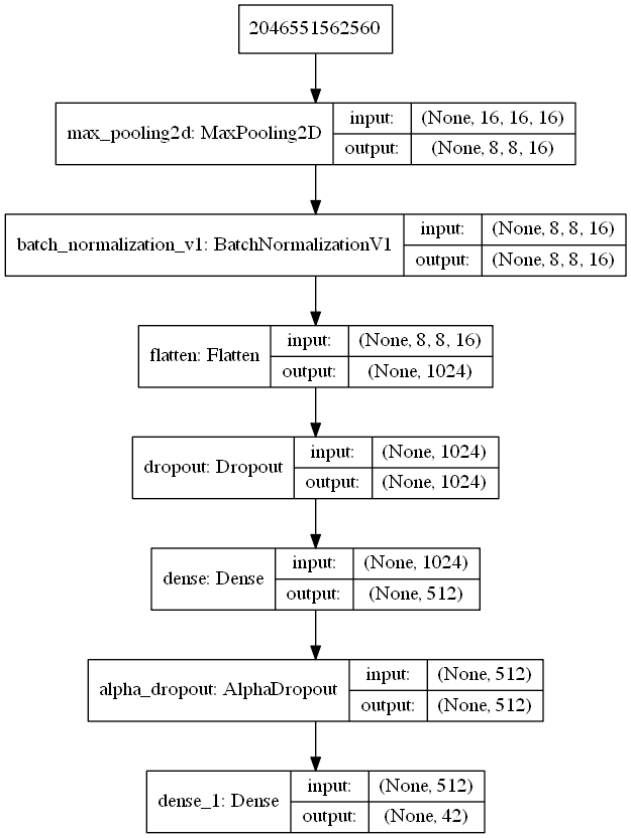

Figure 3: Fully-Connected Classifier Design

5. Experiments

5.1. SELU versus ReLU with BatchNorm in Autoencoder

SELU, as proposed in [4], was investigated to be activation function in our models. Compared to ReLU which is widely used in other neural network models, it has advantageous properties such as avoidance of vanishing gradients and self-normalization effect.

To compare the performance between the two, an experiment was carried out based on our autoencoder model. For fairness, Batch Normalization layers were added to the case of ReLU to provide normalization effect.

After 10 epochs *adam lr 1e-3, batch 32*	ReLU-BatchNorm *Adam b2 0.999*	SELU *Adam b2 0.99*
Train Loss	7.168e-3	**6.565e-3**
Val. Loss	7.078e-3	**6.218e-3**
Train Accuracy	0.5838	**0.6719**
Val. Accuracy	0.5953	**0.6656**
Time Taken	**8m9s**	9m12s

Table 4: Results of Experiment 5.1

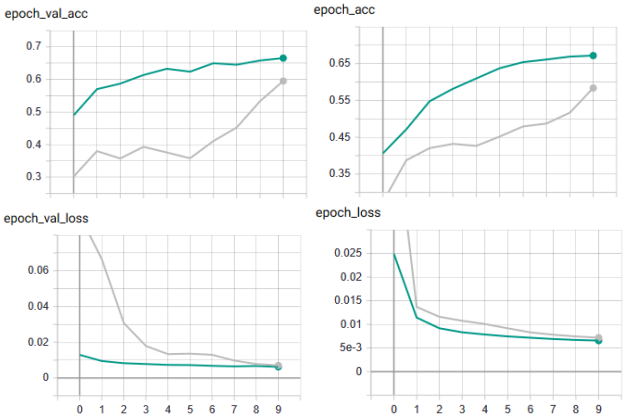

Figure 5: Metrics collected during Experiment 5.1
Green: SELU, Grey: ReLU-BatchNorm

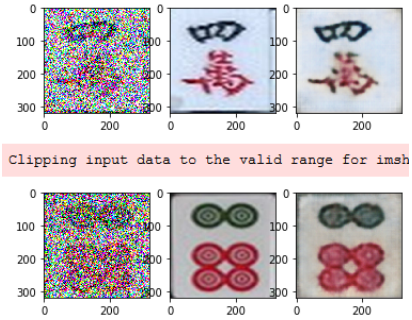

Figure 6: Output images of ReLU-BatchNorm
Left: noisy input, Middle: clear input, Right: output

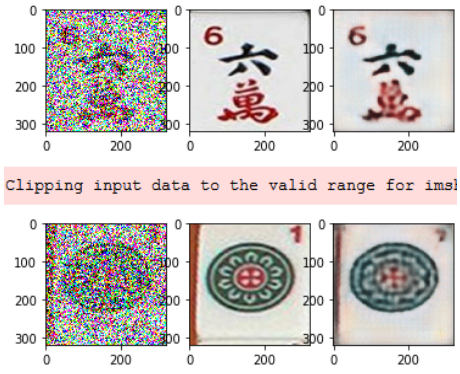

Figure 7: Output images of SELU
Left: noisy input, Middle: clear input, Right: output

From the results, we can see that SELU yielded lower loss and higher accuracy than ReLU with Batch Normalization, albeit taking longer processing time. Also, there is no noticeable visual quality difference between output images from the two configurations. Therefore, SELU was chosen as the activation function in our models.

5.2. Learning Rate for Autoencoder

Multiple learning rate candidates were tested for the autoencoder model with Adam optimizer used.

After 5 epochs *adam, batch 32*	1e-3	3e-3	5e-3	7e-3
Train Loss	8.125e-3	**6.996e-3**	**6.982e-3**	9.827e-3
Val. Loss	7.597e-3	**6.189e-3**	7.176e-3	0.01436
Train Acc.	0.5574	**0.6659**	0.6522	0.47
Val. Acc.	0.5874	0.6557	**0.6908**	0.4398
Time Taken	4m3s	4m3s	4m1s	4m12s

Table 8: Results of Experiment 5.2

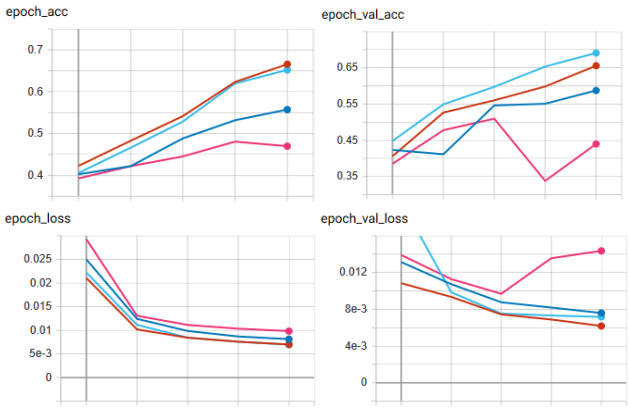

Figure 9: Metrics collected during Experiment 5.2
Dark Blue: 1e-3, Brown: 3e-3, Light Blue: 5e-3, Pink: 7e-3

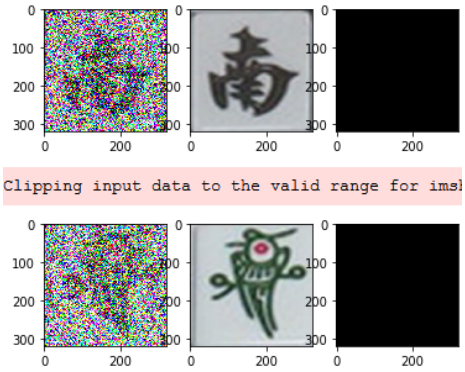

Figure 10: Output images of 9e-2
Left: noisy input, Middle: clear input, Right: output

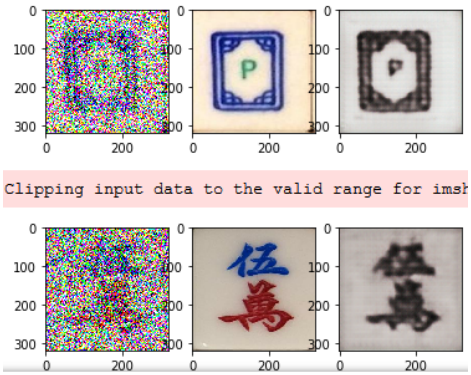

Figure 11: Output images of 7e-3
Left: noisy input, Middle: clear input, Right: output

It is shown that 3e-3 and 5e-3 yielded similar training loss, while 3e-3 gave lower validation loss between the two. Thus, 3e-3 was chosen for the autoencoder model.

One interesting point to note is that when the learning rate is set too high (as in 9e-2), the produced image is a block of single colour (black as shown above or other colour such as yellow during undocumented trials). Whereas when the learning rate is set too low (as in 7e-3), the produced image is a monochromic.

5.3. Number of Hidden Layers in Classifier

For the classifier architecture, different variations on number of hidden layers with 512 neurons have been experimented with.

After 30 epochs *adam lr 5e-4, batch 32, dropout off*	1 layer	2 layers	3 layers
Train Loss	**0.0262**	0.0344	0.07948
Val. Loss	**1.544**	3.057	3.124
Train Accuracy	**0.9943**	0.9926	0.9807
Val. Accuracy	**0.7159**	0.5767	0.6136
Time Taken	2m11s	2m10s	2m11s

Table 12: Results of Experiment 5.3

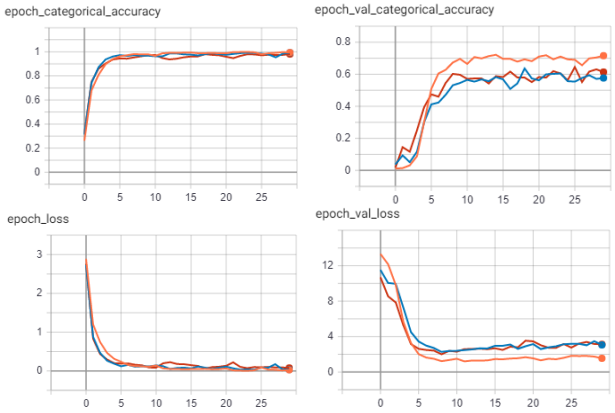

Figure 13: Metrics collected during Experiment 5.3
Orange: 1 layer, Blue: 2 layers, Red: 3 layers

It can be seen that single hidden layer design performed much better in validation, while the multilayer designs performed similarly to each other with worse results. All designs performed similarly in training. This suggests that one hidden layer is sufficient to model the data without overfitting the training set. Therefore, single hidden layer design was chosen for the classifier model.

5.4. Size of Hidden Layers in Classifier

Based on the single hidden layer design, 512 and 1024 neurons were tested as layer size candidates.

After 30 epochs *adam lr 5e-4, batch 32*	512 units	1024 units
Train Loss	**0.01927**	0.0373
Val. Loss	**1.88**	2.561
Train Accuracy	**0.9966**	0.9915
Val. Accuracy	**0.6676**	0.5994
Time Taken	2m12s	2m13s

Table 14: Results of Experiment 5.4

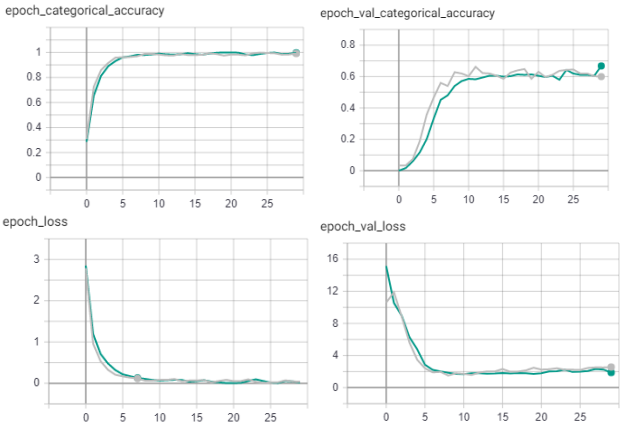

Figure 15: Metrics collected during Experiment 5.4
Green: 512 units, Grey: 1024 units

It is observed that both layer sizes performed similarly, with 512 units giving better validation loss. This suggest that extra units would only overfit training set. Thus, 512 units was chosen for the design.

5.5. Learning rates of classifier

Different learning rates candidates have been tested with the final design.

After 50 epochs *adam, batch 32, dropout off*	3e-5	5e-5	7e-5
Train Loss	0.155	0.03033	**98581e-3**
Val. Loss	1.211	**1.161**	1.348
Train Acc.	**0.9977**	**1**	**1**
Val. Acc.	0.625	0.6591	**0.696**
Time Taken	5m 30s	5m 29s	5m 33s

Table 16: Results of Experiment 5.5

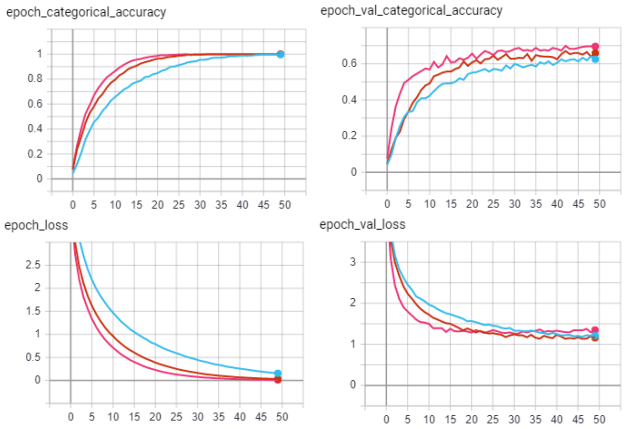

Figure 17: Metrics collected during Experiment 5.5
Light blue: LR 3e-5, Red: LR 5e-5, Pink: LR 7e-5

It is observed that all three candidates overfitted training set while 5e-5 gave best loss during validation. Therefore, 5e-5 will be used in the final model.

5.6. Final Denoising Convolutional Autoencoder

The final autoencoder model is trained for 40 epochs with Adam optimizer with learning rate 3e-3, beta1 0.9, beta2 0.99 and batch size 32.

Best Epoch (40)	Loss	Accuracy
Training	4.4379e-3	0.7587
Validation	4.5001e-3	0.7635
Testing	4.5000e-3	0.7541

Table 18: Results of Experiment 5.6

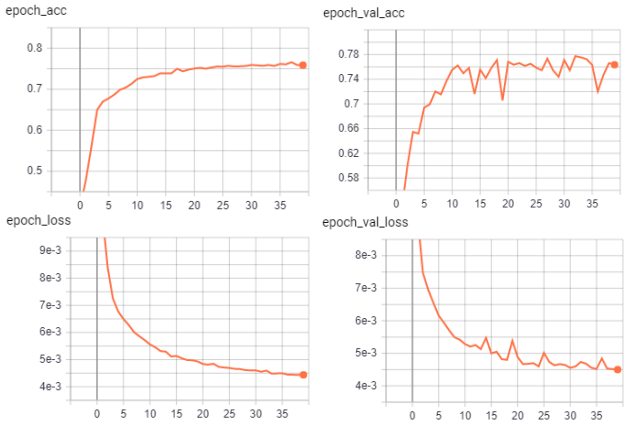

Figure 19: Metrics collected during Experiment 5.6

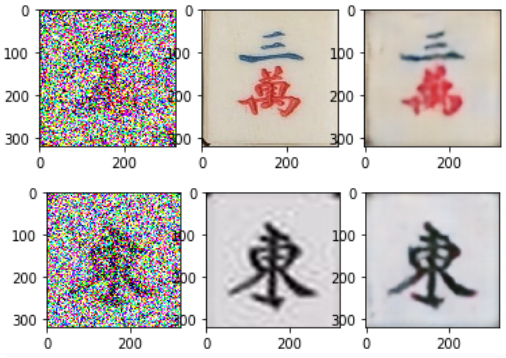

Figure 20: Output images of Experiment 5.6
Left: noisy input, Middle: clear input, Right: output

5.7. Final Classifier Model

The final autoencoder model is trained for 100 epochs with Adam optimizer with learning rate 5e-5, beta1 0.9, beta2 0.99 and batch size 32.

Best Epoch (91)	Loss	Top 1 Acc.	Top 3 Acc.	Top 5 Acc.
Training	0.3455	0.9108	/	/
Validation	1.517	0.6619	/	/
Testing	1.0055	0.7159	0.8921	0.9526

Table 21: Results of Experiment 5.7

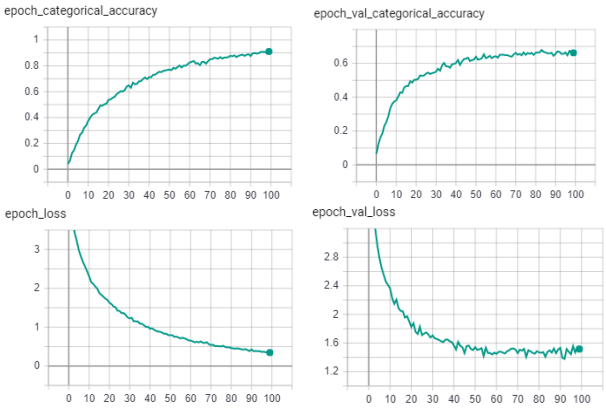

Figure 22: Metrics collected during Experiment 5.7

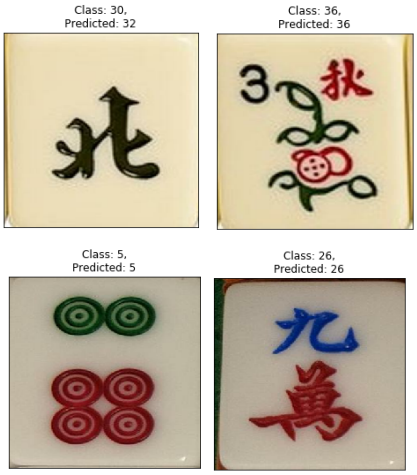

Figure 23: Sample predictions of Experiment 5.7

6. Conclusion

A neural network model was built to classify mahjong tile images by transferring feature encoding layers of a denoising convolutional autoencoder to fully-connected layers. The proposed design reached about 70% accuracy in testing. One major limitation of this project is that the dataset for training is small and the trained model may not generalize well, which could be improved if more samples are to be collected and used. The model may be improved by passing the encoded features to alternative classification methods such as random forest algorithm or

neural networks that are more robust than the proposed fully-connected classifier.

References

[1] Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[2] Vincent, Pascal, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of machine learning research11.Dec (2010): 3371-3408.

[3] Masci, Jonathan, et al. "Stacked convolutional auto-encoders for hierarchical feature extraction." International Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 2011.

[4] Klambauer, Günter, et al. "Self-normalizing neural networks." Advances in neural information processing systems. 2017.