Computer Vision Competition
Game:Rock,Paper,Scissor

peiyan, Chen
School of Computer Science
The University of Adelaide, Australia
a1788396@student.adelaide.edu.au

siyu, Huang
School of Computer Science
The University of Adelaide, Australia
a1810323@student.adelaide.edu.au

Abstract

The dataset contains images of hand gestures from the Rock-Paper-Scissors game. The target is categorizing each hand gesture into three subjects (Rock/ Paper / Scissor). Based on the known information, all the images were taken on a green background with relatively consistent tones and white balance. In this competition, we have conducted experiments based on the given CNN model to achieve the best trade-off between performance and efficiency. For example, the accuracy of the original model and parameters was around 98% with 1.14G Flops. We applied three tricks based on these data and achieved 100% accuracy with 0.29G Flops.

1. Introduction

1.1. Project background

One of the most impressive achievements of the human visual system is its ability to perceive and comprehend the complex visual environment quickly, precisely, and thoroughly [1]. Visual recognition refers to the numerous activities involved in understanding what we see in a visual situation.

Based on the provided PyTorch and convolutional neural network (CNN) models, we aim to recognize the rock-paper-scissors sign gesture on the image. In addition, appropriate modifications were made to the baseline model to improve efficiency and increase the accuracy of the tests.

The convolutional neural network is a deep learning algorithm that takes an input picture, allocates importance (learnable weights and biases) to aspects/objects in the image, and is enabled to distinguish the differences between them.

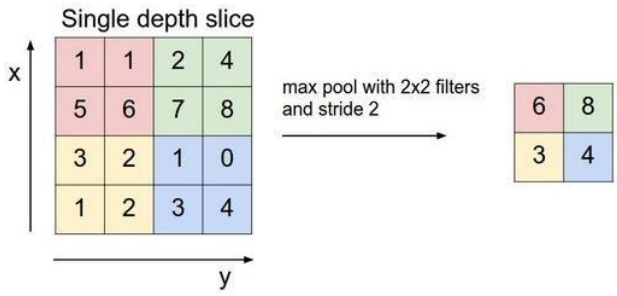
We employed three tricks with ablation studies to improve the accuracy, which are Change of advanced training parameters, Use of a new loss function, and Data augmentation.

The overall strategy uses the concept of the "Greedy algorithm" which always makes the best choice locally for the situation at hand.The locally optimal solutions are then synthesized to achieve the overall optimum. However it does not guarantee 100% that the final solution is optimal, but we will compare the results under different combinations and answer in a dialectical way.

1.2. Understanding of baseline model

Six convolutional layers were utilized in this CNN model (baseline mode) to extract features, and five fully connected layers were used to categorize the features. The convolution kernel size is 3x3 and stride is 1. While larger kernels capture more information about the image, the relatively small size of the kernel reduces computational effort and enhances non-linearity. Stride is the move step of the nearby convolution set to 1, which means that no sample is skipped. Padding and stride are equal, which are both 1. The aim is to make the output size equal to the input size, provided that it is consistent with the stride.

After each convolution there is also a pooling operation, which is used to pool the non-overlapping regions of the image, taking only the largest number in each region. In addition to this, MaxPooling plays an important role in spatial invariance.

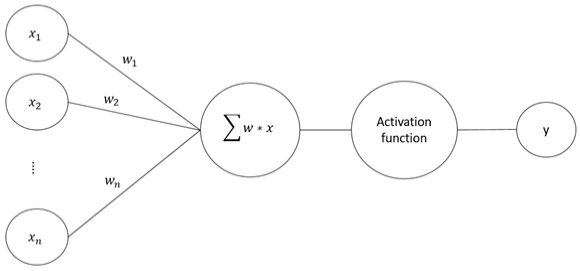


working process of MaxPooling[6]

In the pooling layer, ReLU is more expressive for linear functions of the model. For non-linear functions, ReLU does not suffer from gradient disappearance as the gradient of the non-negative interval is constant, which allows the convergence rate of the model to be maintained in a steady state.

2. Change of advantaged training parameters

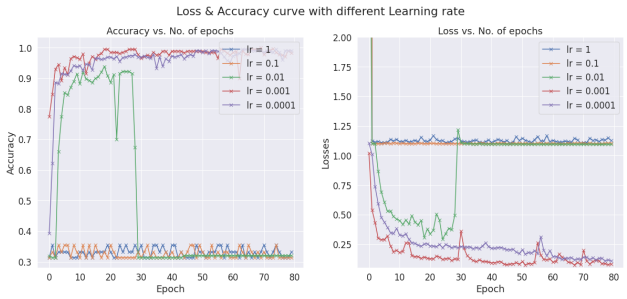
In Deep learning, a model is defined or represented by the model parameters. However, training a model entails selecting the optimal hyperparameters for the learning algorithm to employ in order to learn the best parameters for appropriately mapping input characteristics (independent variables) to labels or targets (dependent variable) and making high performance. In addition, the model does not update the hyperparameters according to the optimisation strategy, it always requires human intervention. For example: Learning rate, batch size, etc.



[Fig.1]
The core skeleton of Neural Networks [2]

2.1. Learning rate

The learning rate is a very important hyper -parameter. When the learning rate is too large it makes the model difficult to converge while if it is too small it will converge too slowly. Only a suitable learning rate will allow the model to converge to a minimum point rather than a local optimum or saddle point.



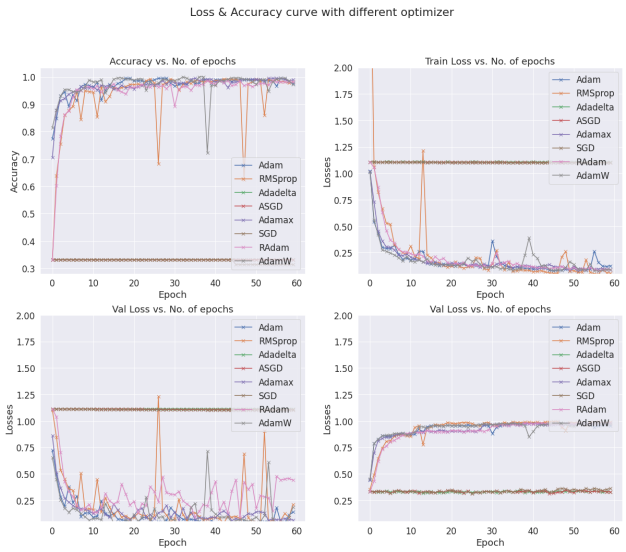
[Fig.2]
result observation of different learn rate

First, a large learning rate is set and the trend and rate of accuracy and loss are observed at that learning rate. The graph above shows the learning rates at 1, 0.1, 0.01, 0.001, and 0.0001 respectively. The conclusion is that the accuracy is highest and convergence is fastest when the learning rate is 0.001. Also, the difference between the output of the model and the true value is the smallest compared to the other cases [Fig.2].

2.2. Optimizer

Optimizer is algorithms or methods used to change neural network properties (Such as weights and learning rates) to reduce losses.

Different optimizers were tested separately, one method being SGD and its modifications (with Momentum); the other being parameter adaptive learning rate methods, including AdaGrad, RMSProp, Adam, etc. Finally, the training loss and validation loss of different optimi- zers are analyzed [Fig. 3].



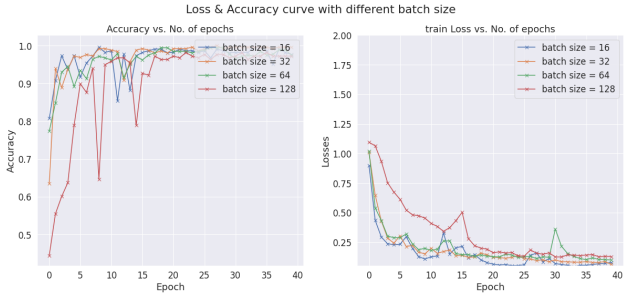
[Fig.3]

test result of accuracy, train loss and val loss

Overall, the validation loss is consistently lower than the training loss. This indicates that the model is acceptable. By comparing in the accuracy charts, we find that the SGD and Adam series optimizers are excelling. Finally, the Adam optimizer is randomly selected for the next stage.

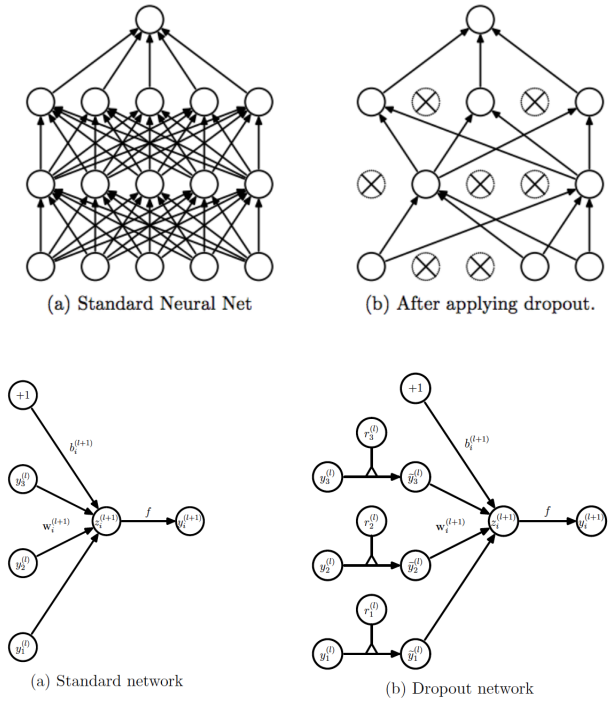
2.3. Batch size

The batch size determines the number of samples that pass through the network at once. According to experimental results, a smaller batch size requires less memory. Since it uses fewer samples to train the network, the whole training process requires less memory and is performed faster. At the same time, the drawback is that the smaller the batch size, it will lead to the estimation of gradient inaccurately.The following figure shows that there is no significant difference in accuracy when the batch size is 32 and 64, and that the flops are independent of the batch size, so we keep the original as the experimental parameter.



2.4. Drop-out

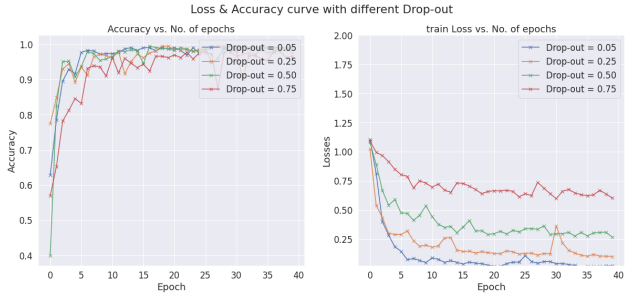
According to the article Multiple Instance Fuzzy Inference Neural Networks, drop-out can effectively prevent model overfitting by dropping the neural network units temporarily.It is clearly observable that 'Tree' becomes sparse after passing drop-out. [Fig.4]



[Fig.4]

Changes in the model brought about by drop-out

In addition, suppose a neural network with N nodes has a drop-out, it can be considered as a set of 2n models, but the number of parameters to be trained is constant and the computational time is a little bit reduced.

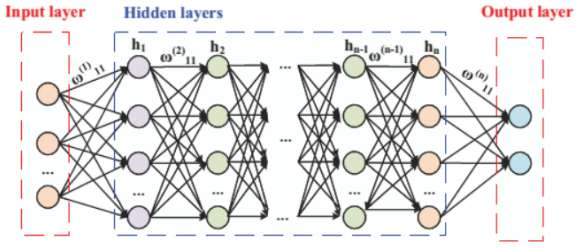


The above graph shows the test results for drop-out of 0.05, 0.25, 0.5, and 0.75. By looking at the line graph, the optimal solution is when drop-out is 0.25. Also, the value of drop-out does not affect the flops.

2.5. Network Structure (Deeper and Wider)

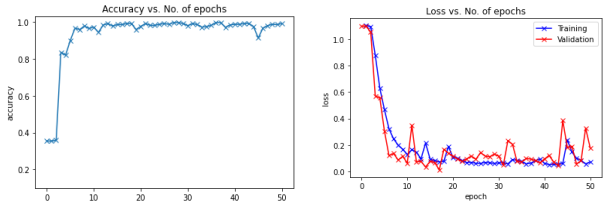
We try the architecture depth and width of the CNN network structure to extend the model to improve the performance.

According to the Computer Vision lecture understanding, wider networks have more sub networks and are more likely to produce gradient coherence than smaller networks, giving them better features and thus better generalization. At the same time, depth gives the network a strong representation capability, which can give the network a strong "learning" capability [Fig.5].

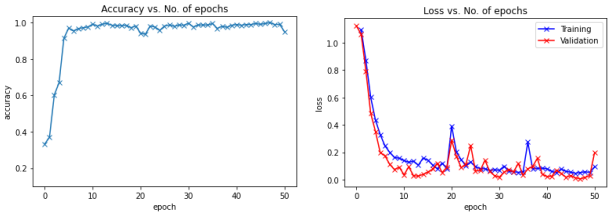


[Fig.5]

The conventional deep neural network architecture



VS



[Fig.6]

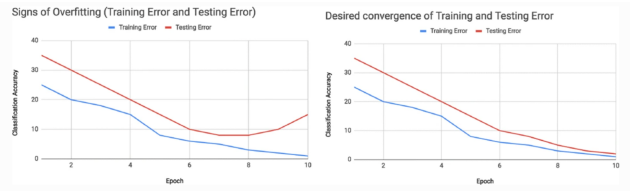
For the experimental observations, I tested the deepening, widening, and combined widening and deepening modes separately. By observing the data results, the widening and deepening shown in [Fig.6] network did not improve the accuracy much but rather slowed down the program. I think this is partly because the Rock-Paper-Scissor dataset is too small and the learned features are not good.

As a result, the original CNN network structure is best for this dataset and should not need to modify it.

3. Data Augmentation

3.1. Motivation

Deep convolutional neural networks have performed remarkably well on image classification tasks. But still, they are heavily reliant on big data to avoid overfitting. The graph below depicts what overfitting might look like when visualizing these accuracies over training epochs [Fig. 7].



[Fig.7]

The plot on the left-hand side shows a turning point where the validation error starts to grow as the training rate continues to drop. The increased training resulted in the model

overfitting the training data and hence performing badly on the testing set. In contrast, the plot on the right-hand side shows a model where the validation error continues to decrease with the training error, which is the desired relationship between training and testing error.

Therefore, to build useful Deep Learning models and to prevent overfitting, the validation error must continue to decrease with the training error. Data Augmentation has been found to be a very powerful method of achieving this. A more comprehensive set of data points can be represented by augmented data to minimize the distance between the training and validation set, as well as future testing sets.

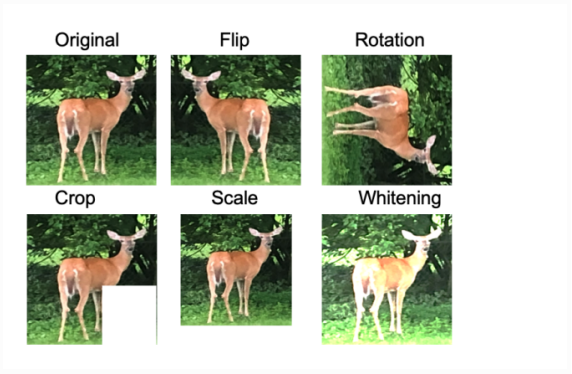
3.2. Prevent Overfitting and increase performance

To prevent overfitting and increase generalization performance, many strategies have focused on the model's architectures. This has led to a sequence of progressively more complex architectures from AlexNet to VGG-16, ResNet, Inception-V3, and DenseNet. Other functional solutions such as drop-out regularization, batch normalization, transfer learning, and pre-training are also developed to extend Deep Learning for application on several smaller datasets.

In contrast with the techniques mentioned above, Data Augmentation approaches overfitting and generalization performance from the root of the problem, the training dataset. A very strong assumption is made here: more useful information can be extracted from the original dataset through augmentations.

There are various data augmentation techniques have been investigated for transforming images into other types of images using them as if they were new data. Geometric transformation techniques such as flipping, cropping, rotation, and translation can artificially modify the geometry, position, and shape features of objects in an image with simple operations, allowing the generation of new features of objects. In addition to geometric transformation, changing the lighting conditions and sharpness of images can artificially produce new features of objects. These methods have the advantage that the data augmentation can be performed with simple image manipulations. However, the disadvantage is that human inspection is required, because changes in geometry or lighting can cause the objects in an image to lose their original features (Shorten and Khoshgoftaar, 2019).

Representative examples are shown in Figure 8.



[Fig.8]
Representative images with data augmentation techniques, such as flipping, rotating, cropping, scaling, and whitening. Data augmentation techniques are used to generate more training data.)

3.3. Applying Variety of Data Augmentations

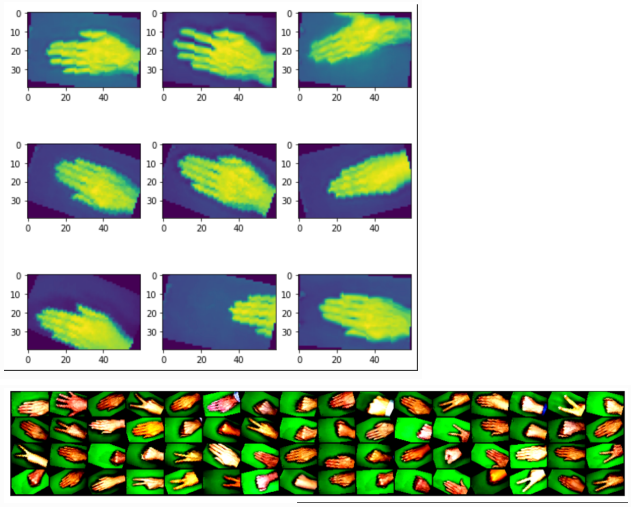
For the hand gesture classification task, four data augmentation techniques: random rotation,

random erasing, Grayscale and Flipping were investigated.

3.3.1. Data Augmentation with Random Rotation

Random rotation refers to a data augmentation technique that can randomly rotate a source image clockwise or counterclockwise by some number of degrees, changing the position of the object in the frame.

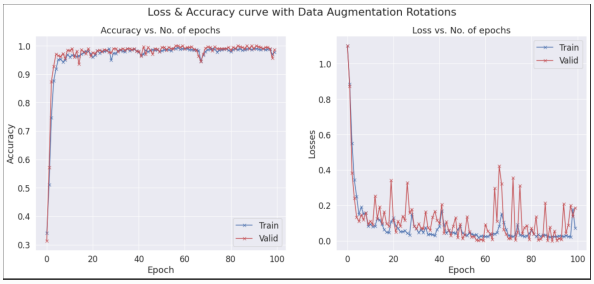
The [Fig.9] demonstrates the effect of random rotation on a set of hand gestures:



[Fig.9]

One observation has been drawn from the above examples: To perform random rotations, the image must either have its corners cut off on the top and bottom or have the image increase in size to avoid cropping edges.

The figure below [Fig.10] shows the result obtained by applying random rotation on the hand gesture image classification task.



[Fig. 10]

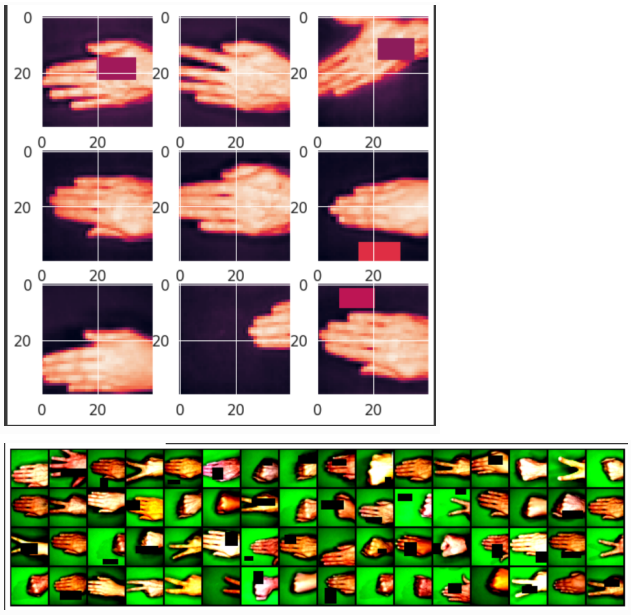
The testing accuracy has achieved 90% after 5 epochs and continues to increase, and the highest test accuracy is 100%. The testing loss shows noisy movements around the training loss which indicate that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize.

3.3.2. Data Augmentation with Random Erasing

Random erasing is another interesting Data Augmentation technique developed by Zhong et al. Inspired by the mechanisms of drop-out regularization, random erasing can be seen as analogous to drop-out except in the input data space rather than embedded into the network architecture. This technique was specifically designed to combat image recognition challenges due to occlusion. Occlusion refers to when some parts of the object are unclear. Random erasing will stop this by forcing the model to learn more descriptive features about an image, preventing it from overfitting to a certain visual feature in the image. Aside from the visual challenge of occlusion, in particular, random erasing is a promising technique to guarantee a network pays attention to the entire image, rather than just a subset of it. Random erasing works by randomly selecting an $n \times m$ patch of an image and masking it with either 0 s, 255 s, mean pixel values, or random values. Random erasing is a Data Augmentation method that seeks to directly prevent overfitting

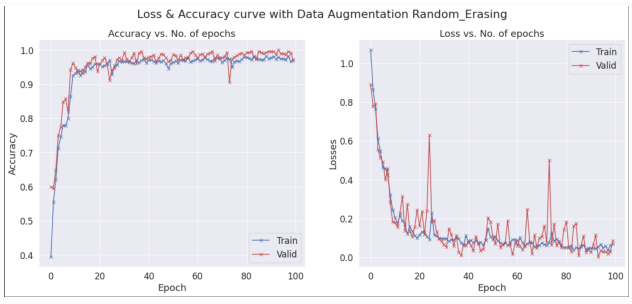
by altering the input space. By removing certain input patches, the model is forced to find other descriptive characteristics. This augmentation method can also be stacked on top of other augmentation techniques such as horizontal flipping or color filters.

The figures below [Fig.11] demonstrates the effect of random erasing on a set of hand gestures:



[Fig.11]

The figure 12 shows the result obtained by applying random erasing on hand gesture image classification.



[Fig.12]

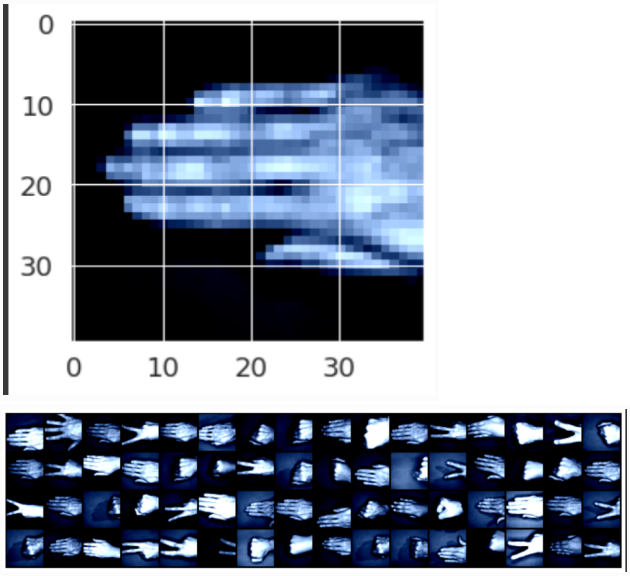
Observations on the results:

Similar to the previous results, the plot shows the model seems to have converged. The highest testing accuracy has also achieved 100%. However, the testing loss shows a larger gap between training loss.

3.3.3. Data Augmentation with Grayscale

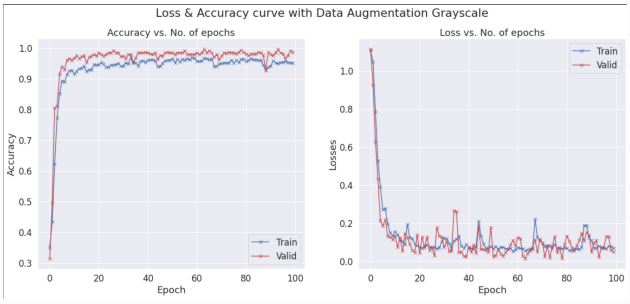
Grayscale augmentation causes an input image to be converted to a single channel, a grayscale output image. This helps our model to place a lower emphasis on color as a signal which is what we desired.

The figures below [Fig.13] demonstrates the effect of grayscale on a set of hand gestures:



[Fig.13]

The figure 14 shows the result obtained by applying grayscale on hand gesture image classification.



[Fig.14]

Observations on the results:

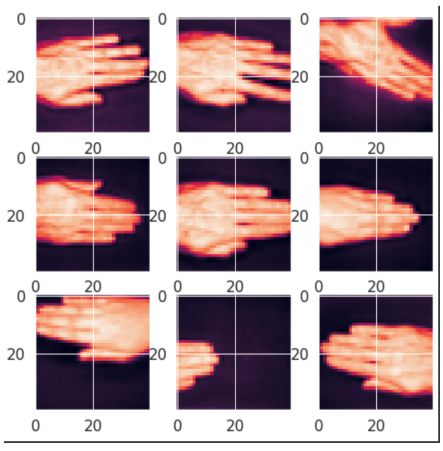
Although the highest testing accuracy didn't achieve 100% when grayscale was applied, the gap between testing loss and training loss has reduced significantly. The learning rate or batch size may be tuned to achieve better smoothness of the loss curve.

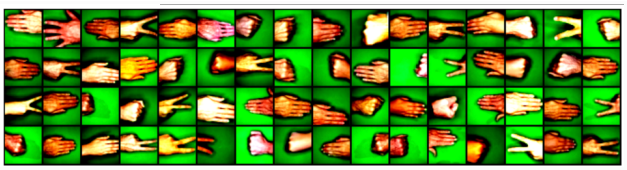
3.3.4. Data Augmentation with Random Flipping

Flipping an image vertically or horizontally is a deceivingly simple technique that can improve model performance in substantial ways.

Our models are learning what collection of pixels and the relationship between those collections of pixels denote an object is in-frame. But convolutional neural networks tend to be quite brittle: they might memorize a specific ordering of pixels that describes an object, but if that same object is mirrored across the image, our models may struggle to recognize it.

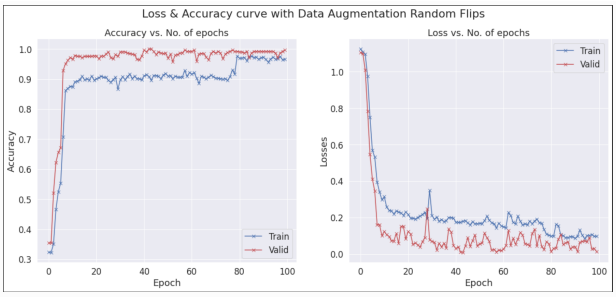
The Figure 15 demonstrates the effect of random flipping on a set of hand gestures:





[Fig.15]

The figure below [Fig.16] shows the result obtained by applying random flipping on hand gesture image classification.



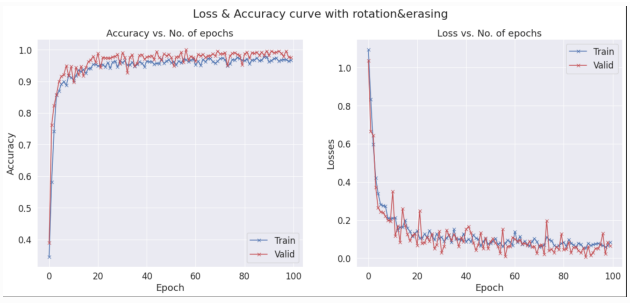
[Fig.16]

Observations on the results:

The plot [Fig.16] shows the model seems to have converged, but the line plot for accuracy and loss shows bumps, and there is a huge gap between them.

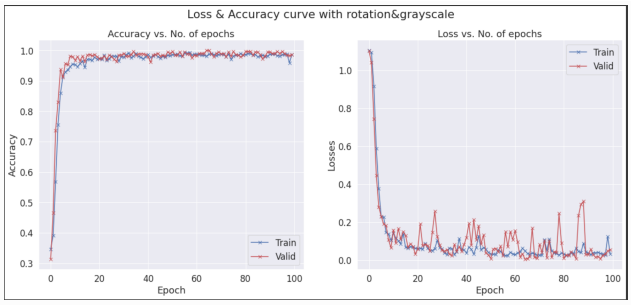
3.3.5. Combinations of Different Data Augmentations

The following plots demonstrated how accuracy and loss curves were influenced by applying different combinations of data augmentations on the dataset.



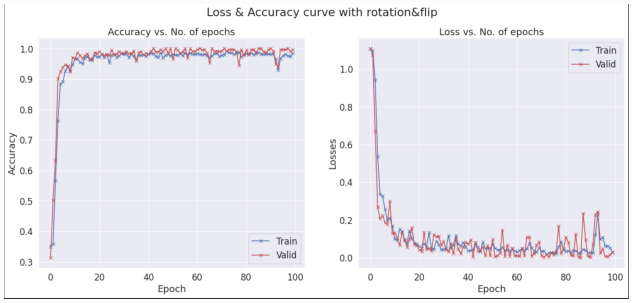
[Fig.17]

The plot [Fig.17] shows the results when rotation and erasing were applied. Both accuracy and loss curve seem to be converged. The loss curve shows some bumps, but is a huge improvement in comparison to the baseline.



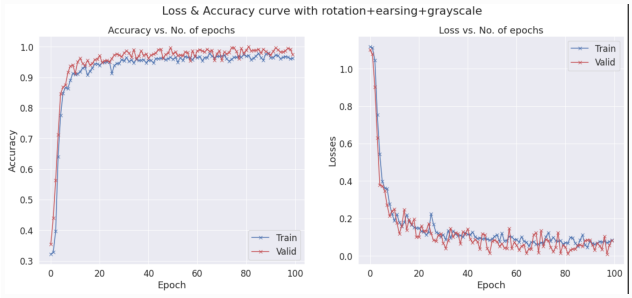
[Fig.18]

The plot [Fig.18] shows the results when rotation and grayscale were applied. The accuracy curve seems to be converged whereas the testing loss curve fluctuates a lot around the training loss curve.



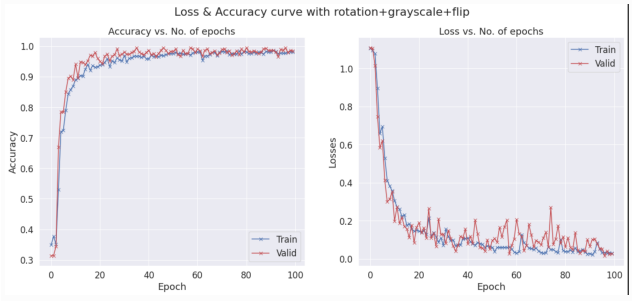
[Fig.19]

The plot [Fig.19] shows the results when rotation and flip were applied. Both accuracy and loss curve seem to be converged. The testing loss curve shows a huge fluctuation around the tail.



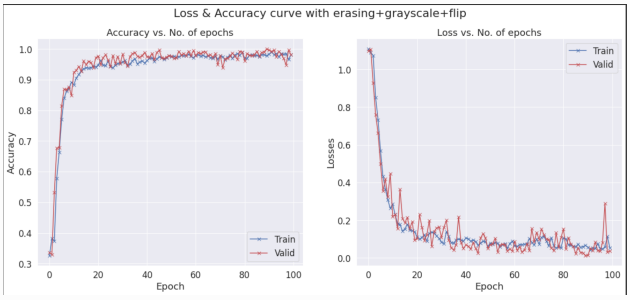
[Fig.20]

The plot [Fig.20] shows the results when rotation, erasing and grayscale were applied. Both accuracy and loss curve shows that the model has converged, although some fluctuations and bumps were shown.



[Fig.21]

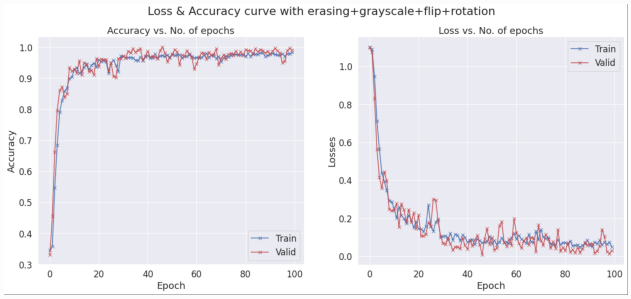
The plot [Fig.21] shows the results when rotation, grayscale and flip were applied. The plot suggests that the model has converged. The loss curve is not as smooth as the previous loss curves which indicates this combination of data augmentations should have second thoughts.



[Fig.22]

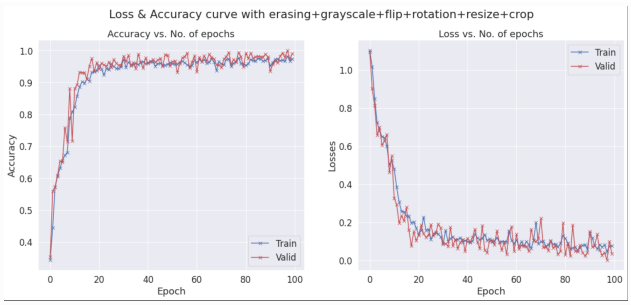
The plot [Fig.22] shows the results when erasing, grayscale and flip were applied. Both accuracy and loss curve shows the model has

converged. Similar to the previous results, the testing loss curve shows some fluctuations, indicating the issue of overfitting cannot be eliminated completely.



[Fig.23]

The plot [Fig.23] shows the result when erasing, grayscale, flip and rotation were applied. The accuracy and loss curve is more smooth than the previous results.



[Fig.24]

The plot [Fig.24] shows the result when erasing, grayscale, flip, rotation, resize and crop were applied. The accuracy and loss curve shows some fluctuations, but converged in the end.

3.3.6. A comparative analysis

Table1: Overall accuracy and loss obtained using different combinations of data augmentations.

Rotation	Erasing	Grayscale	Flipping	Accuracy	Loss
✓				98.7%	0.0562
	✓			99.6%	0.1867
		✓		98.7%	0.0862
			✓	97.8%	0.0560
✓	✓			99.6%	0.0261
✓		✓		99.6%	0.0361
✓			✓	98.7%	0.0383
	✓	✓		97.8%	0.0289
	✓		✓	99.6%	0.0827
		✓	✓	98.7%	0.0383
✓	✓	✓		97.8%	0.0261
✓	✓		✓	97.8%	0.0283
	✓	✓	✓	99.6%	0.0261
✓		✓	✓	99.6%	0.0196
✓	✓	✓	✓	100%	0.0159

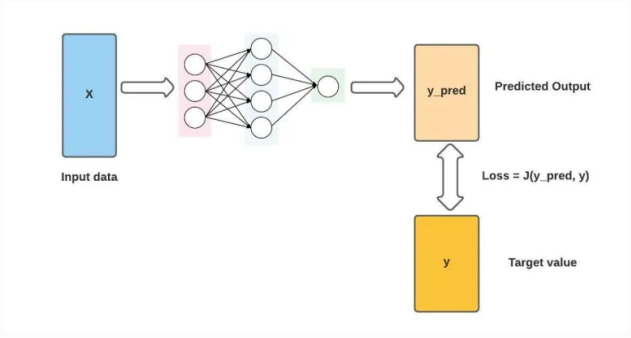
As the comparative results show in Table 1, the highest accuracy and lowest loss were achieved when four data augmentation: Rotation, Erasing, Grayscale and Flipping were applied on the dataset. Therefore, to achieve a better accuracy and to eliminate overfitting as much as possible, we should use Rotation, Erasing, Grayscale and Flipping as our data augmentation.

4. Loss Function

4.1. Motivation

A loss function plays a key role when training our CNN models. It essentially calculates how good the model is at making predictions using a given set of values (i.e., weights and biases). The value being calculated is the loss or error

which denotes the difference between the prediction the model made using a set of parameter values versus the actual ground truth. For example, in our hand gesture image classification task, the loss function is used during training to gauge how well the model can correlate incoming pixels to varying levels of features across the network's hidden layers and to ultimately set the correct probability for each classification. In the hand gesture image classification task, earlier layers could represent basic patterns like curves or shapes, while subsequent layers could start to represent the high-level features of the hand or fingers. At this stage, the loss function plays a role to help the optimizer correctly predict these different levels of features from basic patterns through to the final variety of hand gestures.



[Fig.25]
Working process of Loss Function

Since a loss function guides the whole learning procedure throughout the training, its proper selection is rather important for our CNN model. Thus, the following section will present a comparison of utilizing different loss functions in the context of hand gesture image classification.
Since loss functions generally fall under two categories: classification and regression losses, where classification seeks to predict a value from a finite set of categories, and the goal of regression is to predict a continuous value based on several parameters, we shall only choose the classification loss function for our comparison.

4.2. Applying Variety of Loss Functions

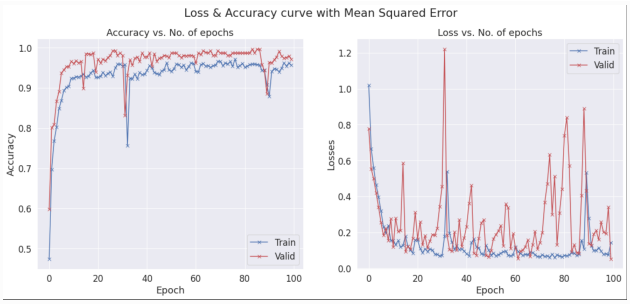
4.2.1 Mean Squared Error

Mean Squared Error(MSE) is one of the basic loss functions where it only takes the difference between the model's predictions and the ground truth, squares it out, and then averages it out across the whole dataset.

Mean Squared Error is expressed as:

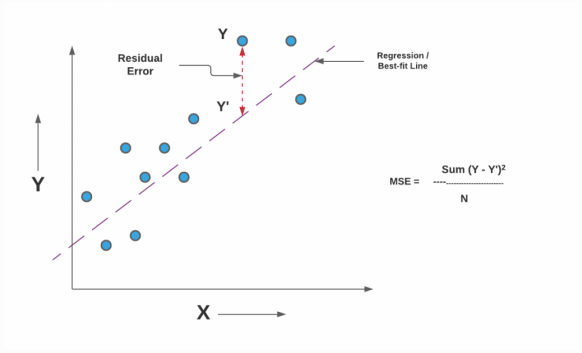
$$\text{loss}(x, y) = (x - y)^2$$

[Fig.26]
MSE expression



[Fig.27]
The difference with two loss function

The plot of Mean Squared Error shows the model has not converged. The fluctuation of the loss curve is a clear sign of overfitting.



[Fig.28]
Show MSE in plot

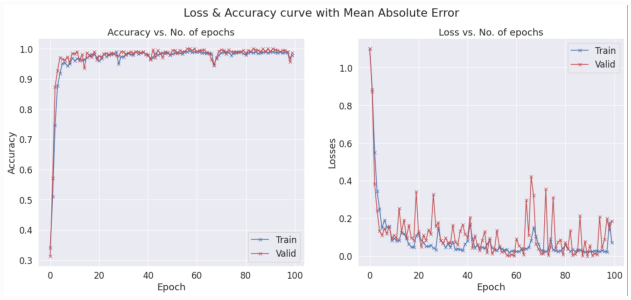
4.2.2 Mean Absolute Error

Mean Absolute Error(MAE) has a similar definition to the MSE, but provides almost exactly opposite properties. To calculate the MAE, we need to take the difference between the model's predictions and the ground truth, apply the absolute value to that difference, and then average it out across the whole dataset.

Mean Absolute Error is expressed as:

$$\text{loss}(x, y) = |x - y|$$

[Fig.29]
MAE expression



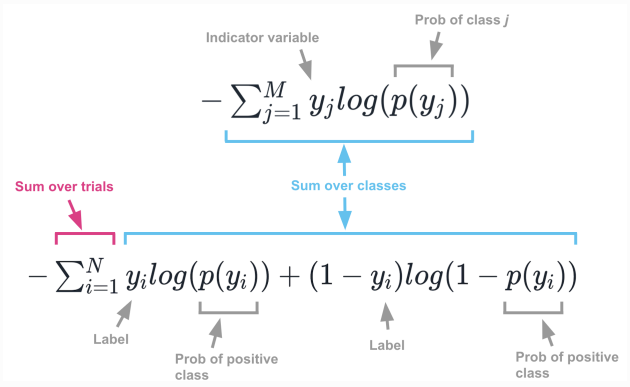
[Fig.30]

The plot [Fig.30] of Mean Absolute Error shows the model didn't converge. Similar to Mean Squared Error, the loss curve shows a huge fluctuation which is a sign of overfitting.

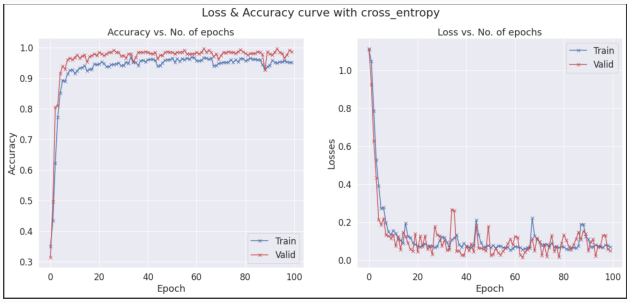
4.2.3 Cross-Entropy Loss

Cross-Entropy(log loss) calculates the difference between the predicted class probabilities and those from ground truth across a logarithmic scale which can be very helpful for image classification.

Cross-Entropy Loss is expressed as:



[Fig.31]



[Fig.32]

[Figure.32] shows that the model has converged. Although the shape of the accuracy and loss curve still has some bumps, it is much smoother than previous results.

4.2.4 Hinge Embedding Loss

Hinge Embedding Loss Function is used for computing the loss when there is an input tensor x and a label tensor y. The target values

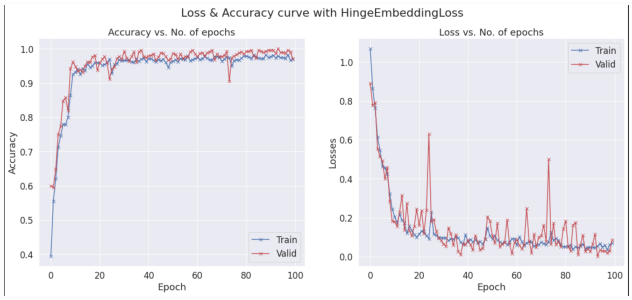
will always be between +1 and -1 which makes it perfect for binary classification tasks.

Hinge Embedding Loss is expressed as:

$$\text{loss}(x, y) = \begin{cases} x \ , & \text{if } y \ = 1, \\ \max\{0, \Delta - x \ \}, & \text{if } y \ = -1, \end{cases}$$

[Fig.33]

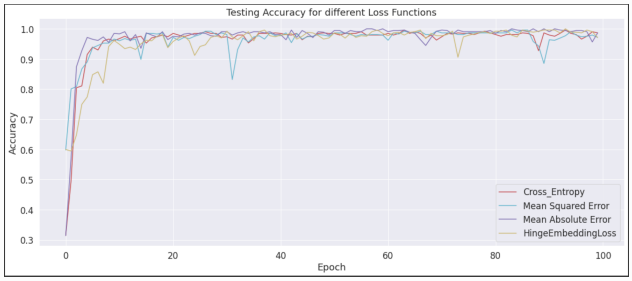
Hinge Embedding expression



[Fig.34]

The plot [Fig.34] of Hinge Embedding Loss shows that the model seems to have converged. However, the loss curve shows a huge fluctuation which is a sign of overfitting.

4.2.4 A comparative analysis



[Fig.35]

[Fig.36]

This section analyzed and compared different loss functions. As the plots shown above, the Cross entropy loss function achieved highest accuracy and ultimately led our model to converge. Therefore, Cross entropy loss function will be chosen over the others.

5. Code Implementation

Our code is not too complicated. For Trick 1, we directly call the different optimizers in the Pytorch. for the other hyperparameters, we take a manual method to test them one by one. For the CNN model, we implemented it according to the method described in Lecture, e.g. adding convolutional and down-sampling layers.

Augmenting the training data is one method for preventing model overfitting and helping the model generalize to a wider range of samples.The hands in every training example are all horizontal and the background is green. What if the hand picture were horizontal instead of vertical? What if the background wasn't so bright? What if the model saw a left hand instead of a right one? We're going to flip, rotate, and change backdrop colors to make our model a little bit more general. The below picture shows our core step of this part.

```
train_transform=transforms.Compose([
        #transforms.RandomHorizontalFlip(),
        #transforms.RandomVerticalFlip(),



        transforms.Grayscale(num_output_channels=3),
        transforms.Resize(40),                  # resize shortest side Hints: larger input size can lead to higher performance
        transforms.CenterCrop(40),              # crop longest side Hints: crop size is usuallt smaller than the resize size
        transforms.RandomRotation(20),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225]),
        transforms.RandomErasing()
```

I take the resize as an example, Larger input sizes can lead to higher performance, which means better recognition. As the functions are wrapped together, we can make the calls directly and just put the arguments we want directly into (). In the next tests, we combine them by the function fits2(). Thus, we can easily choose the most efficient combination. For example: rotation & grayscale, "rotation & flip" and etc.

In addition, We have defined several functions to visualize the results.The function structure is define figure size, import data, set title, comment, and chart legend.

6. Experiment result

Three Tricks were being applied in this competition dataset 1. In [Table 1], the accuracy rates after applying the different Tricks are shown separately in the form of controlled variables.

- Trick1: Change of advanced parameters
- Trick2: New Loss function
- Trick3: Data augmentation

epoch	Trick 1	Trick 2	Trick 3	Accuracy	Flops
5	N	N	N	89.2%	1.14G
100	Y	N	N	97.55%	1.14G
100	N	Y	N	97.55%	1.14G
82	Y	Y	Y	100%	0.29G

[Table.2]

There is a significant improvement in accuracy and flops after applying different tricks. When no optimization is done, the accuracy is around 90% when epoch is 5. By testing different tricks, the accuracy has improved significantly. As shown in table 1 above, when epoch is 100, the accuracy of the experiment has increased by 7-8% compared to before. We tried to optimize again using the new loss function, but did not fetch success, which means that the original Loss function is currently optimal.When we apply all the tricks, we finish with 100% accuracy when the epoch is 82. At the same time, there is a significant drop in Flops which is 0.29G. In addition, the rest of the tricks have no effect on flops except data augmentation.

7. Conclusion

Overall, the classification of the Rock-Paper-Scissor gesture was completed by the method described above. In the course of the experiment, we observed the effect of each parameter on performance according to the Ablation Study and made an optimal decision at each step. As the original model was already running at 98~99% accuracy, the project focused on how to get a higher accuracy rate while achieving high efficiency. In the course of this exploration we made several trade-offs. For example, the VGG model converged very well, but the flops exceeded 20G, far beyond our expectations, so we chose to abandon it. In this competition, we have obtained experience in how to train a CNN model for classifying gestures in images, And it has led to a stronger interest in computer vision. After this competition, we will consider applying more sophisticated NN models to classify sign gestures and see if there is a better solution.

References

[1] L. Fei-Fei, A. Iyer, C. Koch, and P. Perona. *What do we see in a glance of a scene?* Journal of Vision, 2007.

[2] A. Valentina Jul, 2019, *Neural Networks: parameters, hyperparameters and optimization strategies.* Available at : https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5

[3] Z.Zi-bin, Senior Member, IEEE, Y.Yatao, N. Xiangdong , D.Hong-Ning, Z.Yuren. *Wide & Deep*

Convolutional Neural Networks for Electricity-Theft Detection to Secure Smart Grids. Apr 2018.

[4] H.Ping-Han, 2020, *Data Science Dropout*, Available at:https://hackmd.io/@Xg9_wrttQju8FXRCNT-Baw/SJu ahSC3H

[5] F.Hichem, K. Amine ben, *Multiple Instance Fuzzy Inference Neural Networks,* Oct 2016.

[6] R. Vijaya Kumar Reddy, *Effectiveness of Data Augmentation on Handwritten Digit Classification*, Dec 2017

[7] P. J. Huber, “Robust estimation of a location parameter,” Ann. Math. Statist., vol. 35, no. 1, pp. 73–101, 1964.

[8] I. Shendryk, Y. Rist, R. Lucas, P. Thorburn, and C. Ticehurst, “Deep learning - a new approach for multi-label scene classification in planetscope and sentinel-2 imagery,” in IEEE Intl. Geosci. Remote Sens. Symp., 2018, pp. 1116–1119.