Student Number: 221455

1. Introduction

Recently, deep learning models have been quickly introduced and utilized in countless fields as they provide much higher accuracy compared to various existing techniques, including statistical techniques. In particular, recent studies such as computer vision and natural language processing, speech recognition, and robot control, are all based on deep learning due to showing overwhelming accuracy scores. Developing a supervised deep-learning machine network model requires a process of learning, that's why the input value of the model and the correct answer to be output by the model using a large amount of labeled data (Cifar-10) is needed. This model design process consists of several stages and is characterized in that it is repeatedly performed by tuning the model's hyperparameters at each stage through trial and error until satisfactory accuracy is reached. This paper briefly contains the contents of such a series of human-in-the-loop (HITL) processes and the overall procedure of developing a deep neural network(DNN) model to perform multi-class classification.

The developed DNN model consists of a total of five layers. There is one input layer, three hidden layers (convolutional layer), and one output layer that produces results. For preventing overfitting and getting better accuracy and loss scores, hyperparameters are tuned by several training experiments, and Cutmix data augmentation strategy is used. The network was trained through Keras and the final output is the followings: **loss: 1.0191, accuracy: 0.7880, val_loss: 0.5243, val_accuracy: 0.8434**

All of the training processes are explained in this report in the following five sections:

1. Introduction 2. Process of design DNN model 3. Data augmentation - Cutmix 4. Result 5. Appendix (References/All Figures)

2. Process of design DNN model

The aim of this DNN training is to find the exact value of the weight. When there is a layer exists, the transformation that occurs as the data goes through the layer occurs through a function that has the weight of the layer as a hyperparameter. Training means finding weights for all layers to map a given input to the correct output. In addition, in order to get great results from this training process, a well-made network model structure is needed.

A brief summary of implementing a deep neural network model is as follows [2]:

1. **Load and prepare the data (Data processing)**: Input the data set to the program (Cifar-10 images set), and then split the data set into the training data set and test data set (Validation data set). In this paper, two train data sets (1. with Cutmix 2. without Cutmix) are created&used and I compared their performances.
2. **Design the model/Training the data**: With train data set, training the designed DNN model => DNN makes the 'learning algorithm (= Hypothesis)' during the training process.
3. **Evaluate/Tuning the model**: Evaluate the DNN's Hypothesis score (accuracy, loss curve graphs) by the test data set. With the derived scores from the evaluation stage, the model should be tuned and developed to obtain better scores.
4. **Loop**: Return to **3. Evaluate the model** to reevaluate the model and repeat the process of tuning the model to get better output.

2.1. Load and prepare the data (Data processing)

This task configured a DNN using CIFAR-10, an image dataset classified into 10 classes [1]. In the first step, each package and library was loaded to use Keras, TensorFlow, and Cifar-10 data sets, and then the seed (np.random.seed(42), tf.random.set_seed(42)) was fixed. One-hot encoding is also used since the classifier will be a multi-class (10 classes) classifier. After that, loading the pre-shuffled data set and divided into (x_train, y_train), (x_test, y_test) and they were divided by 255, for normalization; single train-val split. I split the images data set into those four groups: x_train(50000, 32, 32, 3), y_train(50000, 10), x_test(10000, 32, 32, 3), y_test(10000, 10)

In this paper, two train datasets(1. using cutmix 2. without cutmix) were created and used, and I compared each output of DNNs. Among them, the first train data was created using data augmentation (cutmix), and the second train data was created in a normal way. Also, the test data was taken by the validation data as it was and used. Therefore, 50,000 images of (32, 32, 3) shape pixels were used as the training data set for learning, and 10,000 images of (32, 32, 3) shape pixels were used as the test data set for evaluating for my DNN model.

2.2. Design the model/Training the data

The process of designing the appropriate DNN model for classification is performed by selecting the appropriate model structure among several models and tuning the model in the direction of showing better results using the accuracy and loss scores obtained from test data. In the step of tuning the model, people normally adjust the hyperparameter value.

The following Figure 1, Figure 6 show the final structure of the DNN model in this paper. This DNN consists of 1 input layer(Convolution), 3 hidden layers(Convolution), and 1 output layer(Dense, Fully connected). In addition, four AveragePooling2D layers(Pooling), five Dropout layers(Dropout), and one Flatten layer(Flatten) are also components of the implemented network model to prevent overfitting and to improve model performance. I implemented the model following the underlying structure of CNN (Conv layer-Pooling layer-FC layer) and tuned it.

Input layer: Convolution2D(filters = 64, kernel_size = 3, strides = 1, padding = 'same', activation = tf.keras.layers.LeakyReLU(alpha = 0.1), input_shape = x_train.shape[1:])
1st Hidden layer: Convolution2D(filters = 128, kernel_size = 3, strides = 1, padding = 'same', activation = tf.keras.layers.LeakyReLU(alpha = 0.1))
2nd Hidden layer: Convolution2D(filters = 128, kernel_size = 3, strides = 1, padding = 'same', activation = tf.keras.layers.LeakyReLU(alpha = 0.1))
3rd Hidden layer: Convolution2D(filters = 128, kernel_size = 3, strides = 1, padding = 'same', activation = tf.keras.layers.LeakyReLU(alpha = 0.1))
Output layer: Dense(10, activation = 'softmax', kernel_regularizer = regularizers.l2(0.01))

2.3. Evaluate/Tuning the model - Hyperparameter

The tuning process of the model through multiple training and testing is essential to design a DNN model that derives great accuracy and loss performance scores. [5] The settings of various hyperparameters adjusted during the tuning process are generally gradually improved to optimized values through repeated trial and error by developers. These hyperparameters are parameters set before the training (before optimizing the weights & bias) process and are largely divided into the following two groups: *1) Hyperparameters related to the how* ***the network is trained*** *2) Hyperparameters related to the* ***network structure***

2.3.1 Tuning process: Hyperparameters related to the how the network is trained

There are many hyperparameters exist that affect model training, such as learning rate, batch size, number of epochs, and selection of optimizers. In this paper, I explored the following hyperparameters:

Explored hyperparameters: Batch size, Learning rate, Optimizer, Number of epochs

2.3.2 Tuning process: Hyperparameters related to the network structure

There are many hyperparameters exist that are related to the structure of the model, such as the number of layers (depth), dropout, weight Initialization, the number of hidden layer units, and the type of activation function (ReLU). In this paper, I explored the following hyperparameters:

Explored hyperparameters: Depth, Width (Conv filter numbers), Dropout, Convolutional filter size (kernel size), Pooling, Activation function

2.3.3 Tuning the Hyperparameters

DNN Setting (Epoch:30)	Accuracy	Loss
2 Hidden layers (Width: 64, 128, 128)	75.88%	81.09%
3 Hidden layers (1) ⋆ (Width: 64, 128, 128, 128)	76.42%	73.79%
3 Hidden layers (2) (Width: 32, 64, 128, 128)	75.55%	76.18%
3 Hidden layers (3) (Width: 32, 64, 64, 128)	74.06%	81.14%
4 Hidden layers (Width: 64, 128, 128, 128, 128)	76.41%	75.71%
Dropout = 0.2	76.42%	73.79%
Dropout = 0.1	77.52%	71.70%
Dropout = 0.05 ⋆	78.83%	66.73%
kernel_size = 2	76.67%	76.05%
kernel_size = 3 ⋆	79.41%	64.89%
kernel_size = 4	79.00%	66.75%
batch_size = 64	79.70%	64.96%
batch_size = 32	79.16%	64.82%
batch_size = 16 ⋆	79.16%	65.02%
With Max Pooling	79.40%	67.09%
Without Max Pooling	74.75%	141.18%
With Average Pooling ⋆	80.50%	63.66%
Act func: Tanh	69.87%	96.90%
Act func: Relu	79.79%	65.46%
Act func: Leaky Relu ⋆	81.91%	59.43%
Learning Rate = 1e-4	65.09%	108.74%
Learning Rate = 1e-3 ⋆	80.89%	62.90%
Learning Rate = 1e-2	23.19%	938.23%
Cutmix (Adam) ⋆	80.65%	63.98%
Cutmix (Rmsprop)	79.98%	68.06%
None Cutmix (Adam)	77.78%	91.53%
None Cutmix (Rmsprop)	77.46%	89.98%

I explored only one hyperparameter at a time by fixing all others to a reasonable arbitrary value. The table above contains the outputs (accuracy&loss score) obtained by training each DNN model used the explored hyperparameter settings by 30 epochs. In particular, for the set value selected as the hyperparameter of the final version DNN model, there is a star mark(⋆) next to it.

For example, I compared the results by setting only the values of dropout to 0.2, 0.1, and 0.05, respectively, with all other hyperparameter values the same. Then, the hyper-

parameter value of the final DNN model is selected as the value of the setting which shows the best performance. e.g. 'dropout = 0.05' shows the best output among 2, 1, 0.05, so this value selected.

However, in particular, for batch_size, a value with no best output was selected. Because, the case of DNN that has 64, 32, and 16 batch_size shows almost similar results of the output. However, the training time shows a difference. (Batch_size = 64: 1321.539 sec, Batch_size = 32, 1142.863 sec, Batch_size = 16, 1096.323 sec)[Describes in code section 3.6] The 'batch_size = 16' DNN model shows the smallest training time. Therefore, I included 'batch_size = 16' value in the final version of the DNN model.

Followings are the learning curve graphs based on each setting of hyperparameter indicates above(accuracy curves and loss curves on training and validation data): [Figure 7, Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, Figure 16, Figure 17; All these graphs are also on the implemented code file.]

The table above indicates that the performance of the selected setting values (with star $\star$ mark) is getting better because the hyperparameters are organized in the order in which they are explored and tuned. Each model has all the hyperparameter values that have been tuned in the previous process. The followings are selected values:

Related to the how the network is trained: Batch size(16), Optimizer(Adam), Learning rate(1e-3)
Related to the network structure: Dropout(0.05), Width(input: 64, hidden 1: 128, hidden 2: 128, hidden 3: 128), Average Pooling, Kernel size(3), Act Function(Leaky Relu)

Moreover, in this task, I defined a model which takes (None, 32, 32, 3) input and predicts (None, 10) output with probabilities for all classes (10 classes). This is why I choose the Dense layer as an output layer for this model.

When training the normal cifar-10 dataset (no data augmentation) with this selected hyperparameter setting, it can be seen from the output graph that overfitting is occurring. The graph (Figure 3) describes that the accuracy score increases as the number of epochs increases, and the validation *loss score* also increases significantly. Therefore, I decided to apply the data augmentation strategy, which is a different method to reduce overfitting, to the DNN model.

3. Data augmentation - Cutmix

Collecting more examples should be the first step in all data science tasks, as more data increases the accuracy of the model while reducing the overfitting. Since one of the main reasons for overfitting is the lack of data to train the network. Apart from normalization, another effective way to cope with overfitting is 'Data augmentation'. It is the process of artificially creating more images from images people already have by changing the size and direction of the image. There are several ways for data augmentation, and I selected '**Cutmix**' strategy for this experiment.

When people classify an object, they sometimes just use only the characteristic parts of the object. Using the regional dropout(eg. Random erasing, Cutout) strategy on DNN, the network can improve generalization performance. However, there is a problem that appears here. When a specific part of the image is removed (dropout), this part may remove useful pixels of the model[6].

Cutmix is a data augmentation strategy that has been proposed to overcome the limitations of this regional dropout method [4]. In short, Cutmix does not simply remove pixels of an image but applies the removed area as a patch taken from another image. Localization performance also improves because objects can be estimated only by looking at the partial appearance of the added patch.

$$\widetilde{x} = M \cdot x_A + (1 - M) \cdot x_B \quad (1)$$

$$\widetilde{y} = \lambda \cdot y_A + (1 - \lambda) \cdot y_B \quad (2)$$

The cutmix strategy cuts(crops) and mix two training sets by the above formula to create a new set as train data[3].

Therefore, in this task, a new train dataset called 'train_data_cutMix' was created by cropping part from the other image using the above formula and combining. The following figure is the combined image after applying the Cutmix method: Figure 4. Moreover, to verify the effectiveness of Cutmix data augmentation, I trained the same model through a Cutmix dataset and a normal dataset. Therefore, I confirmed that cutmix data augmentation slightly increases accuracy score (2.87%) and greatly reduces loss scores (27.55%), and prevents overfitting. The results were derived as follows (Optimizer: Adam, Epochs: 30) Figure 15, Figure 16, Figure 17:

Cutmix; Acc: 80.65% / Loss: 63.98%
None Cutmix; Acc: 77.78% / Loss: 91.53%

4. Result

The DNN model can develop accuracy scores and loss scores by tuning hyperparameters. The DNN model with three hidden layers that finally showed the best accuracy and loss score through various experiences and tuning process is the following structure: Figure 1, Figure 6, and the final output come out (200 epoch); **Acc: 84.34% Loss: 52.43%** (Figure 18) From this assessment, I found that it is possible the DNN model can derive good accuracy through proper adjustment of hyperparameters even if the number of hidden layers is limited. I also saw the superior loss-score reduction effect of the data augmentation strategy(Cutmix). It has been confirmed that the retention of a large number of data is a major requirement to prevent overfitting.

References

[1] fchollet. Introduction to keras for engineers, 2020. 1

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*, volume 25. Curran Associates, Inc., 2012. 1

[3] S. Nath. Cutmix data augmentation for image classification, 2021. 3

[4] D. Walawalkar, Z. Shen, Z. Liu, and M. Savvides. Attentive cutmix: An enhanced data augmentation approach for deep learning based image classification, 2020. 3

[5] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020. 2

[6] S. Yun, D. Han, S. Chun, S. Oh, Y. Yoo, and J. Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features, nov 2019. 3

Appendix

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
average_pooling2d (AveragePooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	73856
average_pooling2d_1 (AveragePooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	147584
average_pooling2d_2 (AveragePooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 4, 4, 128)	147584
average_pooling2d_3 (AveragePooling2D)	(None, 2, 2, 128)	0
dropout_3 (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dropout_4 (Dropout)	(None, 512)	0
dense (Dense)	(None, 10)	5130

Total params: 375,946
Trainable params: 375,946
Non-trainable params: 0

Figure 1. Image with the DNN model (1)

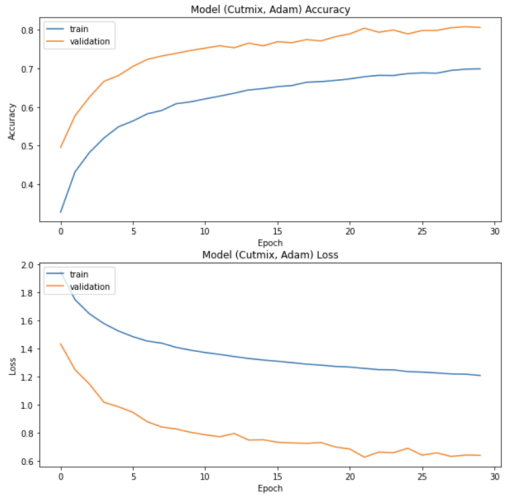

Figure 2. Training the model with 30 epochs(Adam); Cutmix

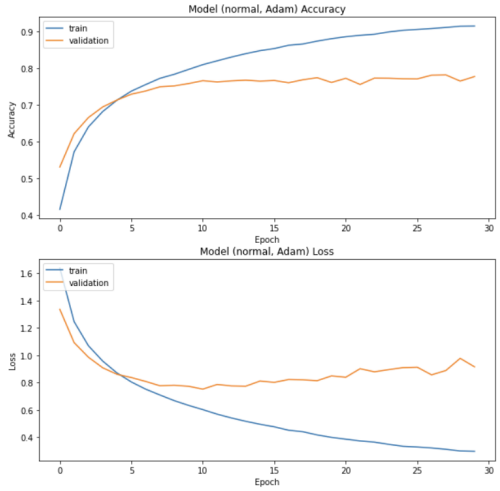

Figure 3. Training the model with 30 epochs(Adam); No Cutmix

Figure 4. Image with Cutmix (Train dataset)

Figure 5. Image with the classification

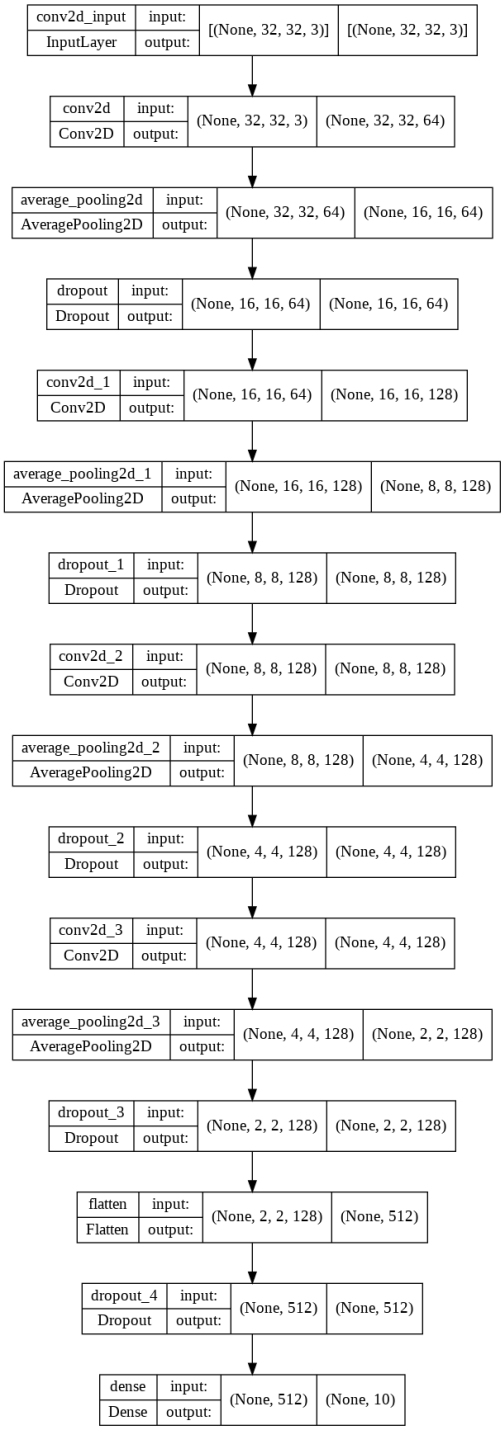

Figure 6. Image with the DNN model (2)

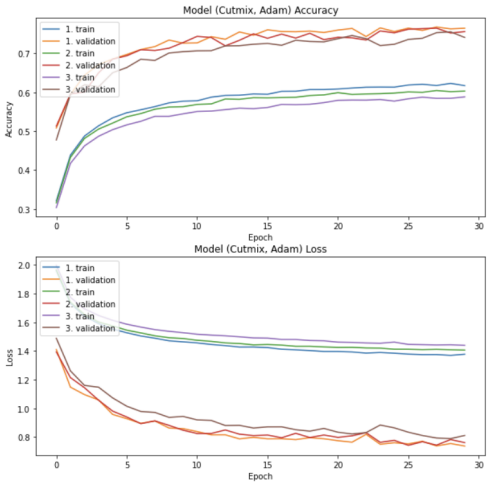

Figure 7. Explored hyperparams: Conv layer's Filter number

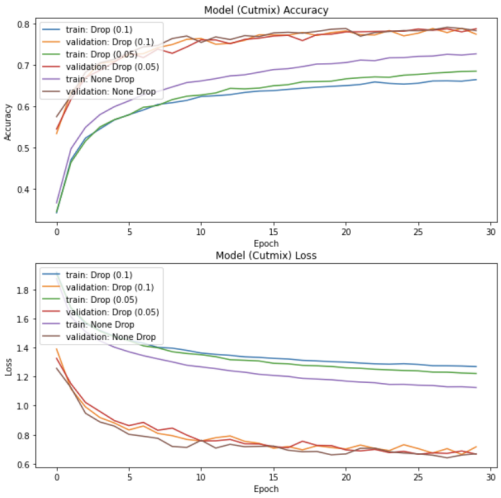

Figure 8. Explored hyperparams: Dropout

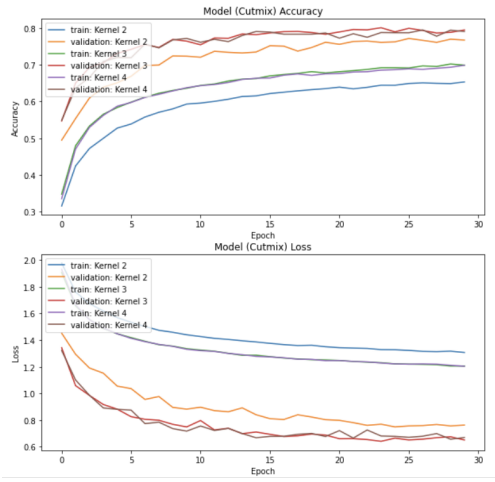

Figure 9. Explored hyperparams: Kernel size

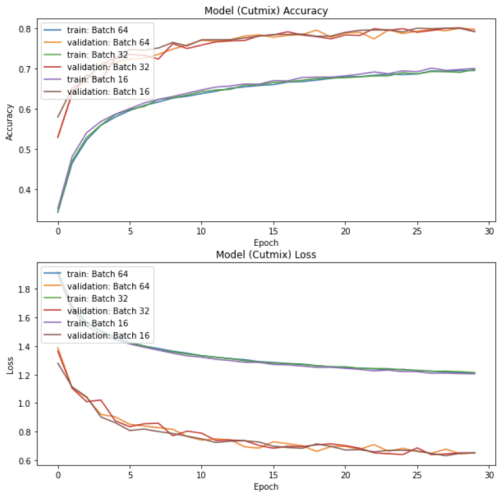

Figure 10. Explored hyperparams: Batch size

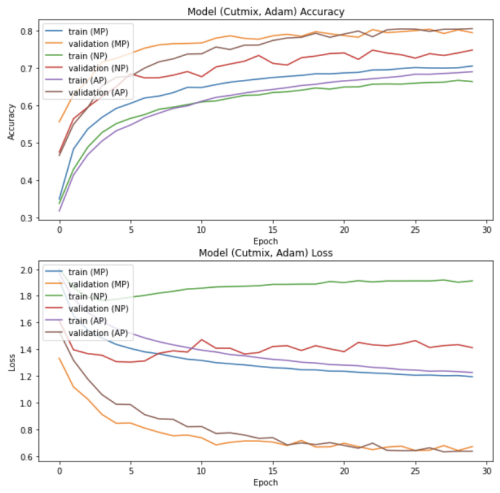

Figure 11. Explored hyperparams: Pooling

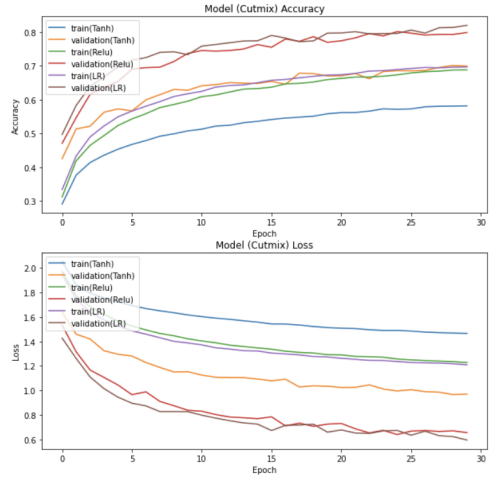

Figure 12. Explored hyperparams: Activation Function

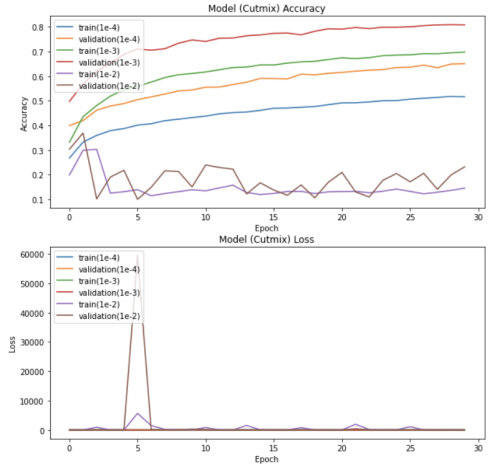

Figure 13. Explored hyperparams: Learning Rate (All)

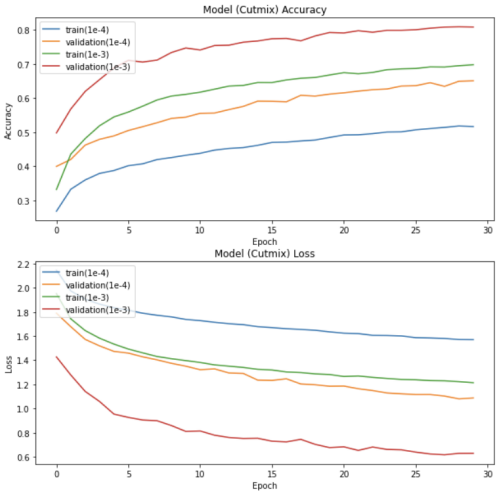

Figure 14. Explored hyperparams: Learning Rate (1e-4 vs 1e-3)

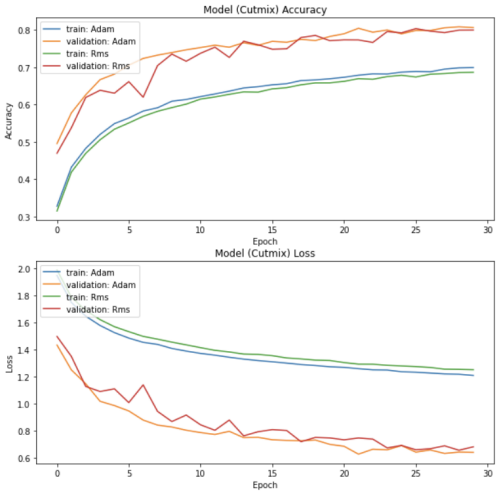

Figure 15. Explored hyperparams: Optimizer (Cutmix)

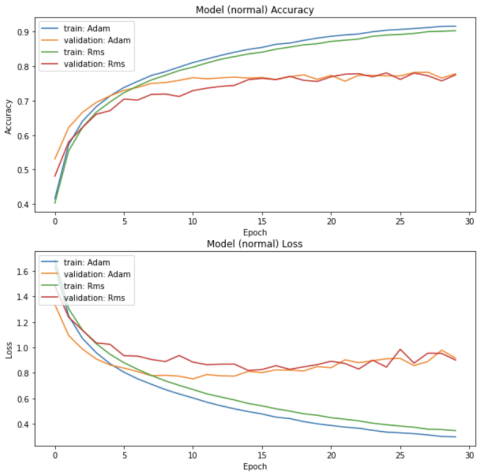

Figure 16. Explored hyperparams: Optimizer (non-Cutmix)

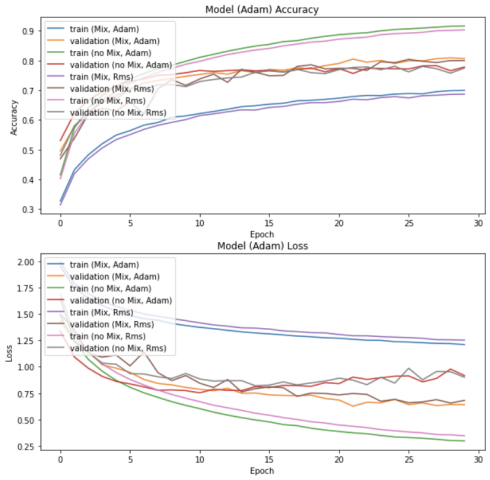

Figure 17. Result of compare the optimizer & cutmix or not

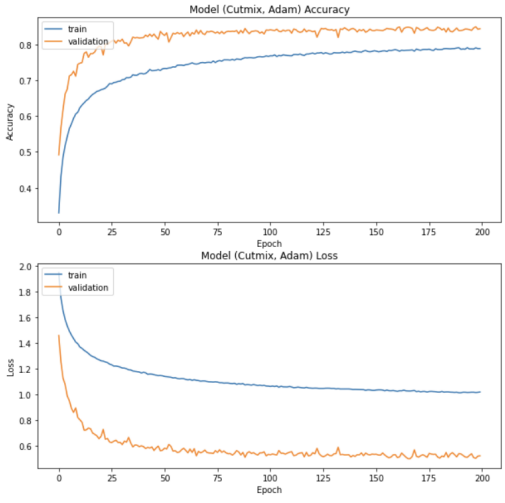

Figure 18. Training the data with the final DNN model (200 epoch/Adam/Cutmix)