A STUDY OF METHODS FOR TRAINING WITH DIFFERENT DATASETS IN IMAGE CLASSIFICATION

Yuxuan Bao

Northfield Mount Hermon School

ABSTRACT

This research developed a training method of Convolutional Neural Network model with multiple datasets to achieve good performance on both datasets. Two different methods of training with two characteristically different datasets with identical categories, one with very clean images and one with real-world data, were proposed and studied. The model used for the study was a neural network derived from ResNet. Mixed training was shown to produce the best accuracies for each dataset when the dataset is mixed into the training set at the highest proportion, and the best combined performance when the real-world dataset was mixed in at a ratio of around 70%. This ratio produced a top-1 combined performance of 63.8% (no mixing produced 30.8%) and a top-3 combined performance of 83.0% (no mixing produced 55.3%). This research also showed that iterative training has a worse combined performance than mixed training due to the issue of fast forgetting.

KEYWORDS

Supervised Learning, Image Classification, Convolutional Neural Network, ResNet, Multiple Datasets

1. INTRODUCTION

Image classification through machine learning can be applied to many different problems, such as product defect checking in factories, product sorting in warehouses, and object identification for everyday use. Currently, many large datasets (for example, ImageNet) are available for use to aid in the training of models to perform image classification. However, there is often great difficulty in using multiple available datasets for the desired categories for a given application, because the different datasets usually have different characteristics that may or may not correspond with the desired application. This research examines the effects of training a convolutional neural network model using two datasets with drastically different characteristics but contain the same categories, using different methods of training and ratios of mixing the datasets, with the goal of developing a method of training with multiple datasets that has good performance on both datasets.

2. NETWORK STRUCTURE

The structure of the neural network model used for this study is a deep neural network with 25 layers, based on the design of the 34-layer ResNet proposed in [1]. The structure is shown in Table 1, while a graphic representation of the network is shown in Figure 1. The details of each of the layers, operations, and blocks, including modifications to the original and rationale for those modifications, are discussed in this section.

2.1. Layers

Layers are the basic building blocks of the neural network model. The model used is very deep and contains many of these layers.

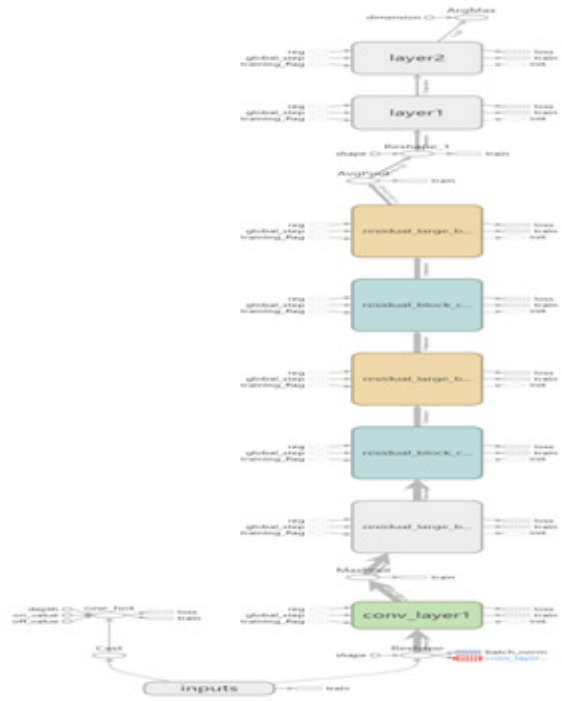


Figure 1. Structure of Entire Network

A visual representation of the entire network. conv_layer is a convolutional layer. residual_large_blocks contain three same-size residual blocks within each. residual_block_changes are changing-size residual blocks. layers are fully connected layers.

Table 1. Network Structure

Layer/Operation	**Parameters**	**Output Size**
Inputs	-	100x100x3
Convolutional Layer	5x5 kernel, depth 3 to 8, stride 1	100x100x8
Max Pooling	2x2 filter	50x50x8
3 Residual Blocks	3x3 kernel, depth 8, stride 1	50x50x8
Change Residual Block	3x3 kernel, depth 8 to 16, stride 2	25x25x16
3 Residual Blocks	3x3 kernel, depth 16, stride 1	25x25x16
Change Residual Block	3x3 kernel, depth 16 to 32, stride 2	12x12x32
3 Residual Blocks	3x3 kernel, depth 32, stride 1	12x12x32
Average Pooling	2x2 filter	6x6x32
Fully Connected Layer	Tanh activation	200
Fully Connected Layer	Softmax activation	26

2.1.1. Fully Connected Layers

Each fully connected layer is a function with multiple inputs and multiple outputs. To obtain the result of one of the outputs, each input is subjected to batch normalization (see Sect. 2.2.1), then multiplied by a weight parameter (different for each input), and then the results are summed together to obtain the preliminary output. A bias parameter, which is a constant, is added to this preliminary, and then an activation function is applied to the result to obtain the final output. Thus, for each output, there are a number of weight parameters matching the number of inputs,

and one single bias parameter. For a layer with m inputs and n outputs, there are m weight parameters and 1 bias parameter for each of the n outputs, and therefore, the entire layer has m x n weights and n biases. For ease of calculation, the weights are stored as an m x n matrix (i.e. 2-dimensional array), and the biases are stored as a 1 x n vector. Thus, the outputs can be easily calculated with matrix multiplication: y = act(xW + b), where W is the weight matrix, **b** is the bias vector, **x** are all the inputs arranged into a vector, **y** are all the outputs in vector form, and act() is the activation function.

Both the weight and bias matrices are trainable parameters; they are changed according to an algorithm every train step. However, they are initialized in a particular way:

Weights are initialized randomly in a truncated normal distribution centered at 0, which is like a normal distribution but guarantees that no values further than two standard deviations from the center will be picked. The standard deviation is set at $1/\sqrt{m}$, where m is the number of inputs. This causes the square deviation of the weights to be inversely proportional to the number of inputs, and thus ensures that the initial preliminary output values are has approximately the same range as the input values. In this network, these values are encouraged to stay between -1 and 1.

All biases are initialized at 0.1. Since these only cause a flat increase to the output value, the number of inputs need not be considered when initializing them. Therefore, all of them are initialized to a constant in order to save time and processing power. However, they could not be initialized at 0—that would cause them to not be trained at all during training, since they would cause no effect on the output values.

This network utilizes two fully connected layers at the end of the network, one with 6*6*32 inputs (from convolutional layers) and 200 outputs, and another with 200 inputs and 26 outputs, with each output corresponding to one classification category. Figure 2 shows the structure of the first fully connected layer, with 1152 inputs and 200 outputs, and using Tanh as the activation function.

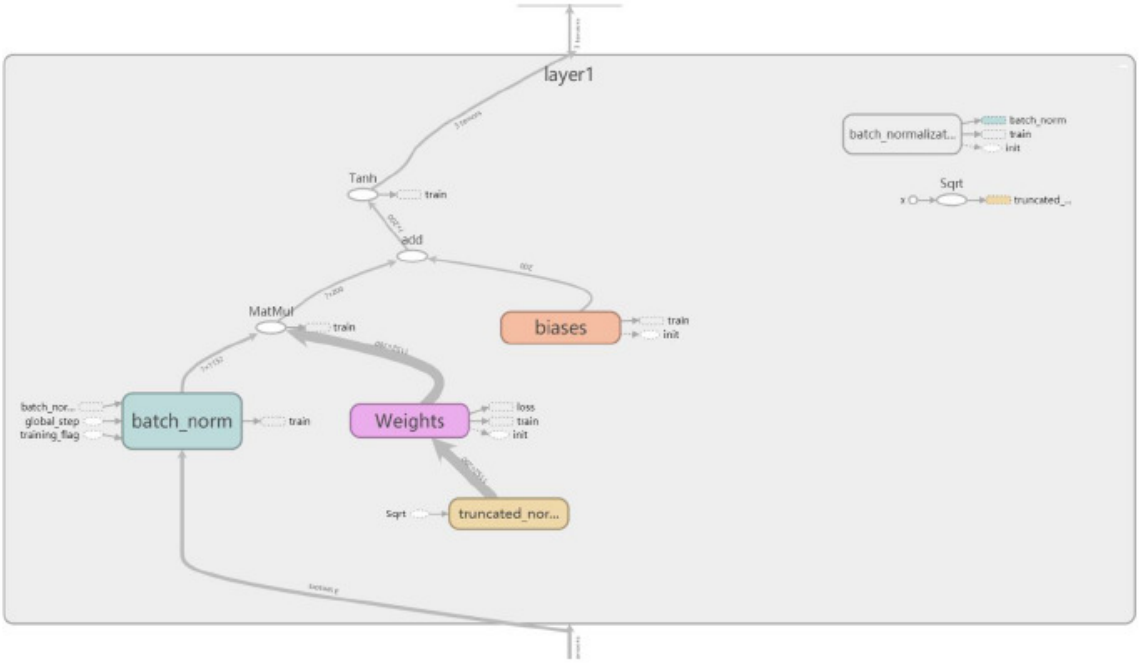


Figure 2. Fully Connected Layer Structure

The fully connected layer is the most versatile layer, since each output is parametrized by every input (hence the name "fully connected"). Therefore, it is the most capable and flexible layer, if resource-intensive. It is utilized for the final layers of the network in order to ensure the flexibility of the network.

2.1.2. Convolutional Layers

Whilst the fully connected layer treats its input as a one-dimensional vector, the convolutional layer treats its input as an image, represented by a three-dimensional tensor. The first two dimensions refer to the regular width and height of the input image, while the third dimension is its depth. For regular images, this refers to their color channels: red, green, and blue. However, intermediate layers may produce images with an arbitrary number of layers which does not have conventional meaning for each layer. In a convolutional layer, the input image is batch normalized (see Sect. 2.2.1), then convolved, then added with bias and affected by an activation function to produce the output. Both the input and the output are three-dimensional tensors.

The central operation performed by a convolutional layer is dependent on a kernel, a four-dimensional trainable parameter. The kernel is a collection of three-dimensional cubes performing operations on portions of the three-dimensional input image. For each such cube, the depth is equal to the depth of the input image, while the width and height are usually much smaller than the original width and height of the input image. The cube slides across the width and height dimensions of the input image, and at each position, calculates the sum of all the entries of the element-wise multiplication to obtain a single number for the preliminary output. As the cube slides across the input image horizontally and vertically, a two-dimensional layer of the preliminary output is formed, with the number at each location representing the result of the calculation when the cube is centered at that location of the input image. For corner cases when the cube extends outside of the input image, such as when the cube is centered at a corner of the image, any point located outside the bounds of the input image is considered to have the value 0 (zero padding). As different cubes slide across the entirety of the input image, different layers of the preliminary output are calculated. Therefore, the number of cubes matches the depth of the output. This entire operation to produce the preliminary output is called a convolution. With respect to TensorFlow specifications, the kernel has dimensions kern_width × kern_height × in_depth × out_depth. Each cube is represented via kern_width × kern_height × in_depth, and there are out_depth cubes. In this network, all convolutional layers use kern_width = kern_height = 3. Since the input image is only 100 by 100, using a 3 by 3 kernel allows the network to learn gradually through layers while reducing the computational cost (smaller kernels contain less parameters to train).

After the preliminary output is thus obtained, a bias parameter is added and an activation function is applied as usual. Notably, the same bias is added for every value in the same layer of the output, which means that there are only out_depth biases for each convolutional layer. This massively reduces the number of trainable parameters while still preserving the complexity of the network.

Since each cube in the kernel is desired to have different parameters, it is important that the kernel weights be initialized randomly as to avoid identical cubes. The weights are initialized in the same way as the fully connected layers, with standard deviation = $1/\sqrt{(\text{in\_depth})}$. (Though the preliminary output values also depend on the kernel width and height, these are always small ($\leq$ 5) in this network and thus are not considered.) The biases are all initialized to 0.1 like the fully connected layers.
The convolutional layer is a basic building block of this network and is thus utilized repeatedly. Figure 3 shows the structure of the first convolutional layer in the network, with a kernel width

and height of 5 and an output depth of 8. The input image has dimensions 100×100×3(from the RGB channels of the original image).

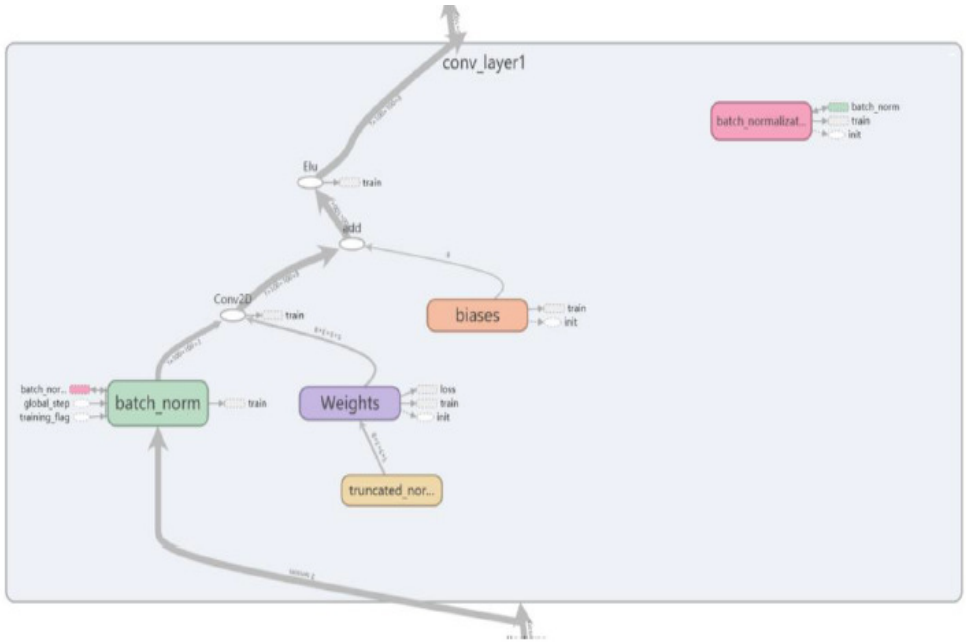


Figure 3. Convolutional Layer Structure

All convolutional layers use the ELU activation function, defined:

$$f(x) = \begin{cases} x, & x \geq 0 \\ e^x - 1, & x < 0 \end{cases}$$

This function has similar benefits as the commonly used ReLU function. Like ReLU, ELU alleviates the vanishing gradient problem by leaving the input unchanged if it is positive. However, there are several additional advantages:

- ELU is differentiable at 0, unlike ReLU. This prevents possible calculation errors while training.
- ELU returns small gradients for negative inputs, while ReLU returns 0. This aids gradient propagation, since gradients are still considered for negative inputs instead of being ignored completely.
- ELU returns negative results for negative inputs, while ReLU returns 0. This preserves the sign of the original value and therefore gives the optimizer more information during training.

The main advantage of the convolutional layer over the fully connected layer is that it drastically reduces the number of trainable parameters (weights and biases), making the network able to be trained in a reasonable amount of time. For the example layer presented above, the convolutional layer has 5×5×3×8=600 weights and 8 biases, for a total of 608 trainable parameters. However, if it was to be replaced by an equivalent fully connected layer, with 100×100×3 inputs and 100×100×8 outputs, the number of weights required would be 100×100×3×100×100×8=2.4×10^8 weights and 80000 biases. Thus, the convolutional layer keeps the number of trainable parameters manageable. Another advantage is that the convolutional layer preserves the spatial relationships on the horizontal and vertical axes of the input image. Similar to how images are normally perceived, pixels that are spatially close are grouped together and considered simultaneously by

the kernel. This should help the network in learning to identify smaller parts of the picture and assembling these parts to determine a classification result.

2.2. OPERATIONS

There are a few operations that are applied within or in-between layers that are explained in detail here.

2.2.1. Batch Normalization

Batch normalization is an operation that involves a set of images, and is performed on each image at the same time in each fully connected or convolutional layer. The goal of this operation is to rescale and shift all the input values so that all values have a mean close to 0 and a variance close to 1.

During each training step, the mean and variance values of the current batch of images are calculated, and used to train two parameters: the moving mean and moving variance. These two parameters are updated during every training step with respect to a hyperparameter bn_momentum, using this formula:

$$\text{moving\_mean} := \text{moving\_mean} \times \text{bn\_momentum} + \text{cur\_mean} \times (1 - \text{bn\_momentum})$$
$$\text{moving\_variance} := \text{moving\_variance} \times \text{bn\_momentum} + \text{cur\_variance} \times (1 - \text{bn\_momentum})$$

A higher bn_momentum causes the current calculated mean and variance to be less impactful on the trained mean and variance. The bn_momentum used in this study increases as the current iteration increases, in an exponential relationship:

$$\text{bn\_momentum} = 1 - \max(0.01, 0.9^{\text{cur\_iter}/50})$$

During both training and testing, each value in every image is adjusted using the trained moving mean and moving variance to make the mean of all values close to 0 and variance of values close to 1. A simple shift and scaling is used to accomplish this:

$$y = \frac{x - \text{moving\_mean}}{\text{moving\_variance}}$$

This operation bounds all values in a range reasonably close to 0 so that training is efficient and meaningful. This is especially important since the ELU activation function is used for convolutional layers, which may cause values to become unbounded on the positive side. In addition, ELU provides the most desirable behavior near zero (the function behavior changes at 0), so this operation bounds all values in a small range near 0.

2.2.2. Pooling

The pooling operation is performed after some convolutional layers in order to reduce the image size of the output, reducing computational cost and condensing data. All pooling layers used in this network have a filter size of 2x2 and stride of 2, so this operation takes a three-dimensional image, splits each layer into 2x2 patches, and then merges each of these patches into a single value, effectively cutting both the height and width of the original image in half, leaving the depth unchanged. There are two ways to merge: average pooling takes the average of the four values to replace them, while max pooling takes the maximum of the four. Following the original ResNet design, this network utilizes a max pooling layer after the first convolutional layer to set up for

the residual blocks and an average pooling layer after the last convolutional layer in order to reduce the computational cost of the succeeding fully connected layer.

2.3. RESIDUAL BLOCKS

Residual blocks are made up of two convolutional layers and a bridge that transfers the input of the first layer to the output of the second layer, adding them together to produce the output of the entire block. The activation function of the second convolutional layer is applied after the original input is added. The central idea is that the network may learn to either use the two layers or to omit them (by learning to push the weights of the two layers to 0). Thus, if the network encounters problems trying to optimize a large network, it may be able to shrink the network and optimize the smaller network instead. For a more detailed discussion of the benefits of residual blocks, see [1].

2.3.1. Same-size Residual Block

The residual block is easier to build when the input of the first convolutional layer has the exact same dimensions as the output of the second convolutional layer, that is, the two convolutional layers do not modify the size of the image. When this is the case, the first input can simply be element-wise added to the second output. Figure 4 shows a same-size residual block with the dimensions of all images at 50×50×8.

2.3.2. Changing-size Residual Block

Building a residual block is trickier when the convolutional layer changes the dimensions of the input image. Since the dimensions of the first input does not match that of the second output, they cannot be directly added together. Instead, a special convolutional layer is added to transform the first input to the proper dimensions. This special convolutional layer uses a kernel with dimensions $1 \times 1 \times \text{in\_depth} \times \text{out\_depth}$, and uses a stride equal to the stride of the convolutional layer that changed the image's dimensions. This will allow the output for this special convolutional layer to match the second output exactly in dimensions, while mostly preserving the values of the input image. The 1×1 convolutions will produce different linear combinations of the input layers for each output layer, and the stride size will omit data in between the strides to reduce image size. After this layer, the results can be element-wise added to the second output as usual. Figure 5 shows a changing-size residual block with input dimensions 50×50×8 and output dimensions 25×25×16.

3. TRAINING

This section describes the details of how the model is trained. This research uses the Tensorflow framework [3] in Python to train and test the neural network models.

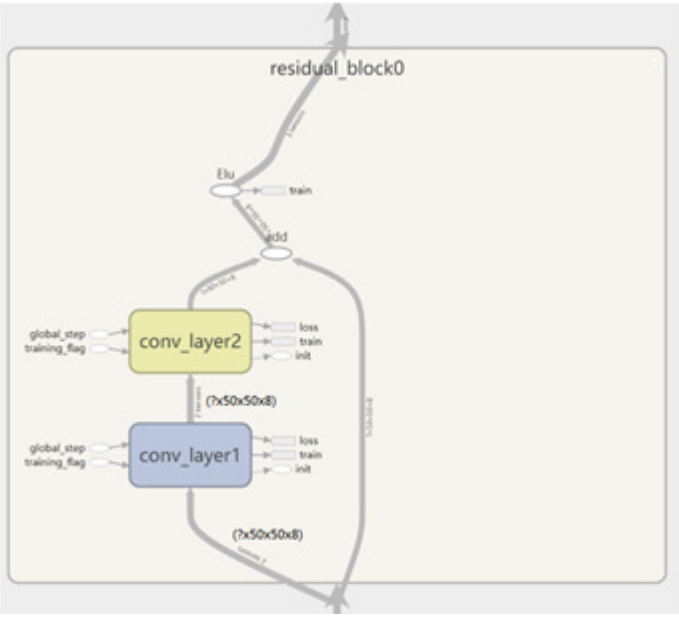


Figure 4. Same-size Residual Block Structure

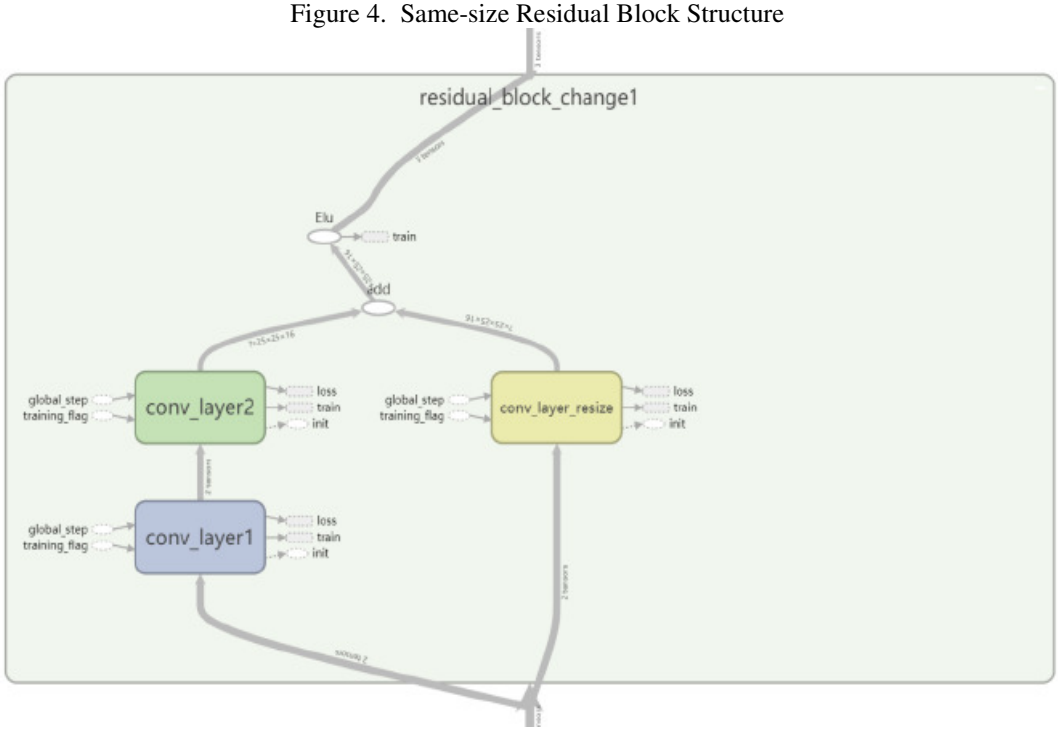


Figure 5. Changing-size Residual Block Structure

3.1 Loss

The loss used for training is the cross entropy, a standard loss for classification tasks. The cross entropy is calculated with err = - y*ln(y_pred), where y is the actual label (0 or 1) of a certain category for a given input image and y_pred is the model's prediction for the probability of that image belonging in that category. The total loss of the entire model is calculated by summing all the errors across the categories for one input image together, then averaging this sum over all input images.

3.2. REGULARIZATION

The model uses L2 regularization, which is added to the total loss for optimization. The regularization factor is tuned as a hyperparameter.

3.3. OPTIMIZER

The optimizer used to minimize total loss is Momentum, which is a variant on the regular gradient descent which moves the weights and biases based on the sum of the gradient and a fraction of the last movement. Therefore, it tries to keep the "momentum" of movement, which could allow the algorithm to skip over some small local minima.

The fraction of the last movement is a hyperparameter, and is called the momentum term. Since the first few iterations of training should allow the algorithm to move in the correct general direction of a minimum, later iterations should try to keep the momentum in that direction. Therefore, for this research, the momentum term depends on the current iteration, and is set to:

$$\text{momentum} = \frac{0.6}{1 + e^{\frac{100 - \text{cur\_iter}}{500}}}$$

A logistic function is used to steadily increase the momentum term as more iterations are performed, but cap the momentum term at 0.6.

3.4. LEARNING RATE

The initial learning rate is tuned as a hyperparameter, but the training process itself changes the learning rate in order to become more precise as training progresses. From the starting learning rate, the learning rate is reduced at each iteration so that it smoothly decays 10% every 80 iterations. In addition, if the program detects that the training loss have not dropped for 500 iterations, then it decides that the training rate must be too high and cuts the training rate in half.

3.5. Batch

Since taking all of the training data available every iteration is very time-consuming, training is done in batches, where each batch of training images is 1/20 of the images in the training set, taken sequentially from the beginning. After running through 20 iterations, the entire training set has been used for training, so it is shuffled and the next iteration use the first 1/20 of the newly shuffled training set.

4. DATASETS

To examine the effects of training with multiple datasets, two datasets with very different characteristics are used. Both datasets contain images of the same 26 categories of fruit, listed here: Apple, Banana, Cantaloupe, Carambola, Cherry, Clementine, Coco, Date, Grape, Grapefruit, Kaki, Kiwi, Lemon, Lychee, Mango, Melon, Orange, Papaya, Passion Fruit, Peach, Pear, Pineapple, Pitahaya Red, Plum, Pomegranate, Strawberry.
Both datasets contain around 400 images per category.

4.1. DATASET 1 (NICE DATASET)

The first dataset is taken from part of the Fruits-360 dataset, a free online database of fruits from [2]. This dataset consists of very nice fruit images. Every image is exactly 100 by 100 pixels wide, has a completely white background, and always contains only one fruit in each image. A typical image from Dataset 1 is shown in Figure 6.

4.2. DATASET 2 (REAL DATASET)

The second dataset is personally gathered as part of the research, and comes from web image searches of the 26 fruit types. In contrast to Dataset 1, this dataset mostly contains rectangular images with non-white backgrounds and may contain multiple fruits in the same picture. Each of these images are cropped into square images (cutting out the edges that are longer, horizontal or vertical), then resized down to 100x100 pixels as part of the program before being fed into the models. However, to keep some things consistent with Dataset 1, all images from Dataset 2 show only the exteriors of each fruit, and will never show a cut-open fruit. A typical image from Dataset 2 is shown in Figure 7.

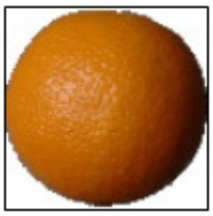


Figure 6. Nice Data Example

Figure 7. Real Data Example

4.3. TEST SETS

One test set is created from each dataset by randomly taking 40% of the images in each category out. This process forms Test Set 1 (Nice Test Set) and Test Set 2(Real Test Set). The performance of models on these datasets is used to evaluate the general accuracy of the models. Whenever a model is tested, it is run on both test sets to give accuracies for each test set, so its performance could be measured for both datasets.

4.4. TRAINING SETS AND TRAINING METHODS

The other 60% of the images from the two datasets form the “nice” and “real” training sets. Images from these training sets are used to train the models. Based on the training method, different sets of images are used from these training sets. The two training methods described in the following subsections are used because they have the advantage of low computational cost. Both methods only require a single model to be trained, and require no additional processing beyond partitioning the dataset randomly. The experiment thus studies the performance of these training methods which only require minimal computational power beyond the base cost of training one model itself.

4.4.1. MIXED TRAINING

In mixed training, a certain mixing percentage x% of the images of the "real" training set and (100-x) % of the images of the "nice" training set are used to train the model. The images are randomly chosen from each training set up to the required percentage, and the chosen images from the two training sets are mixed together to train the models. Since the number of images in the two training sets are around the same, the complimentary percentages ensure an approximately equal amount of training data (around half of all training data) for each mixed training model.

4.4.2. Iterative Training

In iterative training, 50% of the images of the "real" training set are used first to train the model. After that, 50% of the images of the "nice" training set are used to train the last two layers (i.e. the fully connected layers). The incentive for this method of training is that the "nice" training set would provide the model with the general characteristics of each type of fruit (the convolutional layers), while the "real" training set would train the model to identify the learned characteristics at different locations and on different backgrounds (the fully connected layers). Training with the "nice" training set is called initial training, while the secondary training with the "real" dataset is called restricted training.

5. EXPERIMENT AND RESULTS

This section presents the experiments performed on the models and their results. The experiments are designed to test the performance of the two training methods in Sect. 4.4.

5.1. EXPERIMENTAL DESIGN

The performance metric for all models in the experiment is the multiplicative average of the "nice" test set accuracy and the "real" test set accuracy, that is,

$$\text{perf} = \sqrt{acc_n \, acc_r}$$

Multiplication is chosen so that a model is given a better performance score if it performs well on both test sets than if it performs flawlessly on one but badly on the other.

11 different mixed training sets are produced with the mixing percentage at 10% intervals (respectively mixing at 0%, 10%, …, 100% of the "real" training set). Each of these training sets are used to train a different model for a set number of iterations, and these models' hyperparameters are tuned to optimize their performance. Their performances are then compared. The 50% mixed training set is then duplicated for use by iterative training. 9 different models are trained with this exact same training set, training on only the "nice" images first, then on the "real" images. The 9 models have different ratios of initial training iterations to total training iterations, at 10% intervals, from 10% to 90% (0% and 100% are omitted because in those cases either the "nice" or "real" images are not trained on at all, so those are not considered iterative training). The total amount of iterations for each model is constant, and is equal to twice the amount of iterations used for mixed training, because each iteration in iterated training uses half as much data than in mixed training. The iterative models use the best hyperparameters tuned from mixed training.

The performance of mixed training models and iterative training models are then compared to determine the best method of training with two different datasets.

5.2. Hypotheses

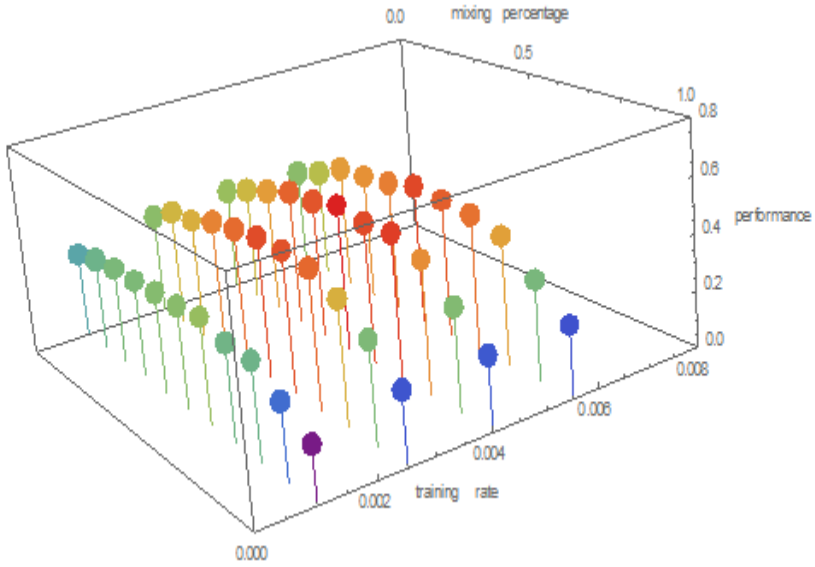
The hypotheses are that 1) in mixed training, the highest test accuracy for each dataset is obtained when the corresponding dataset is mixed into the training set at the highest ratio, 2) in mixed training, the performance score is at the highest near 50% mixing of both datasets, and 3) iterative training produces a better performance score than the best results of mixed training.

5.3. Data and Analysis

5.3.1. Mixed Training Data

Three sets of models are trained to tune the hyperparameters for the mixed training models. The first set is trained to find the best learning rate. Each mixed training set is used to train four

Figure 8. Mixed Training, Learning Rate Tuning Results



models at different learning rates (0.001, 0,0025, 0.004, 0.0055) and the same regularization factor (0.7) for 1500 iterations, and their performance are compared with each other. The results are shown in Figure 8.

The best performance for each mixing percentage is obtained at either a training rate of 0.0025 or 0.004.

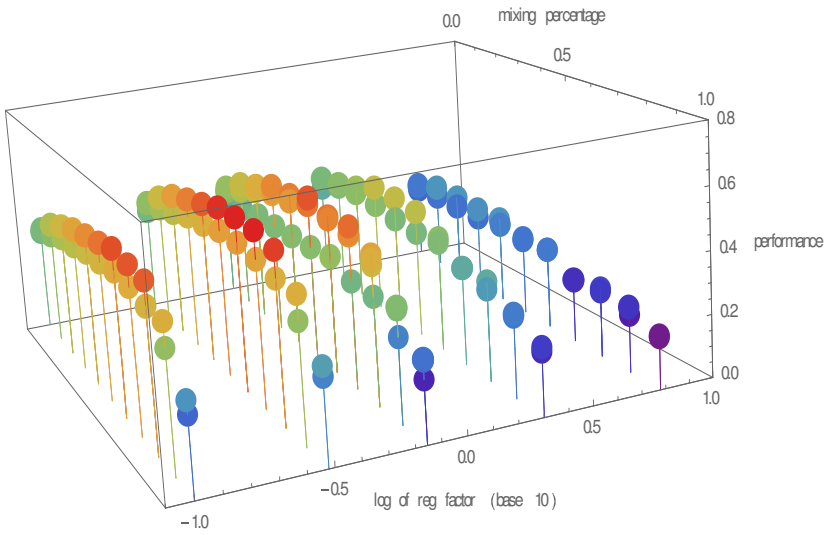


Figure 9. Mixed Training, Regularization Factor Tuning Results

Note that every point shares the same (x, y) coordinate with another point. These represent results of models trained with the same mixing ratio and regularization factor but with different learning rates.

The next set of models is trained to find the optimal regularization factor. Each mixed training set is used to train 8 different models, with all combinations of the two best training rates found and 4 different regularization factors (0.1, 0.3, 2.0, 6.0). They are also trained for 1500 iterations, and their results are combined with the models trained in the first set (at regularization factor 0.7) and compared. The results are shown in Figure 9.

The best performance is obtained for regularization factors between 0.1~0.7.
The final set of models is trained to narrow down the best regularization factor and to finalize the best mixed training model. Since the best performance for the prior sets of models are obtained at the very end of the training process (the 1500th iteration), the improvements in the performances of the models have not stalled out, so the final set of models are trained for 3000 iterations, double the amount of iterations of the previous sets. The models in the final set are trained at a learning rate of 0.004 and regularization factors of 0.1, 0.3, and 0.5. The results are shown in Figure 10.

The set of hyperparameters that produces the best results on the greatest number of cases of mixing percentages is a learning rate of 0.004 and a regularization factor of 0.5. This is the set of parameters used for the experiment.

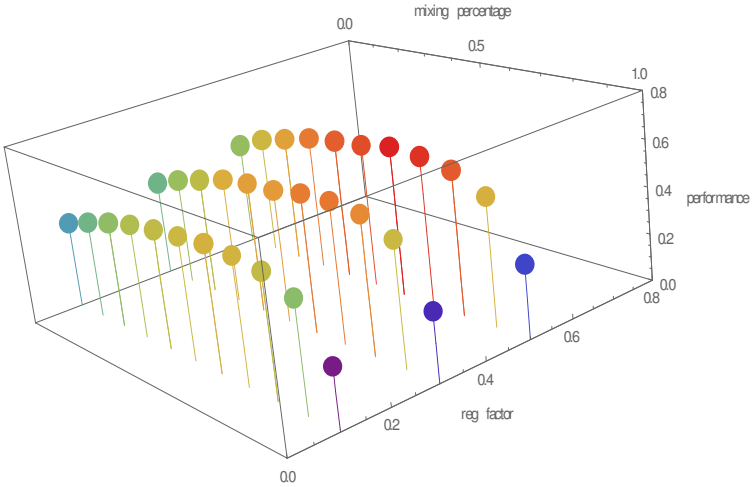


Figure 10. Mixed Training, Final Tuning Results

5.3.2. Mixed Training Analysis

A summary of the highest top-1 accuracies and performance achieved on each mixing percentage is shown in Table 2 and Figure 11.

The results mostly agree with hypothesis 1. The highest accuracy on "real" test data is obtained when the training data is completely "real" data (i.e. a mixing percentage of 100%). Interestingly, however, the highest accuracy on "nice" test data is obtained at a 20% mixing percentage. This implies that partial mixing of the more difficult to learn "real" dataset has a beneficial effect on the test accuracy of the easier "nice" dataset, even though the effect is very slight: the 20% mixing compared to no mixing only increased the accuracy by 0.33%.

Table 2. Mixed Training, Top-1 Accuracies and Performance
Highest accuracies and performance among all mixing percentages are highlighted in orange.

Mixing %	Real Data Accuracy	Nice Data Accuracy	Performance
0	0.22712341	0.9648086	0.468113896
10	0.28601596	0.96574885	0.52549286
20	0.32426092	0.96816653	0.560107852
30	0.36602533	0.9646743	0.593680112
40	0.38925388	0.9638684	0.611715718
50	0.40544346	0.95701814	0.622297615
60	0.44462693	0.94533247	0.648321119
70	0.45283905	0.9002015	0.63847192
80	0.46198967	0.8194762	0.614201621
90	0.47067106	0.6235057	0.540779084
100	0.50093853	0.21517797	0.308003439

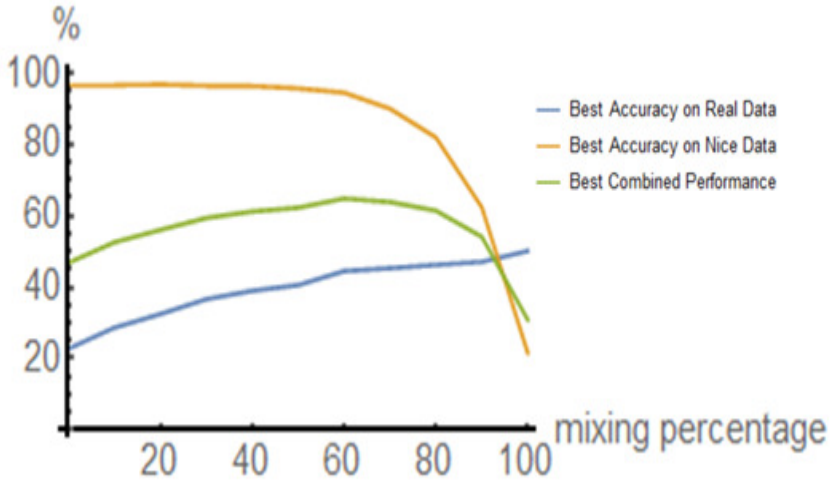


Figure 11. Mixed Training, Summary of Top-1 Accuracies and Performance

The results slightly deviate from hypothesis 2. The performance curve has a noticeable shift towards higher mixing percentage, peaking at 60%. This produces a best top-1 combined performance of 64.8%. The top-3 combined performance curve from Figure 12 shows a similar trend. This curve peaks at a mixing percentage of 70%, which produced a best top-3 combined performance of 83.2%. One possible explanation for this behavior is that the "nice" dataset is easier to learn than the "real" dataset. As the "real" data mixing percentage increases from zero, the "nice" accuracy decreases very slowly at first, showing that the model can learn enough to achieve a near-perfect accuracy on "nice" test data even with a fairly small amount of "nice" training data. Therefore, the combined performance takes more from the trend of the "real" data accuracy curve, which increases as the mixing percentage increases, than that of the "nice" data accuracy curve, so the peak of the combined performance curve shifts right.

Note that whenever the training set only consists of data from one of the datasets (i.e. at mixing percentages of 0% and 100%), the test accuracy of the other data set is 21~23%. If the two datasets were not related at all, then training with only one dataset would cause the model to behave effectively randomly with respect to the other dataset, with a projected accuracy of 1/26≈3.85%, because there are 26 categories in total. However, the actual accuracy being much higher shows that using a dataset as the training set would produce a higher-than-random accuracy on a correlated dataset.

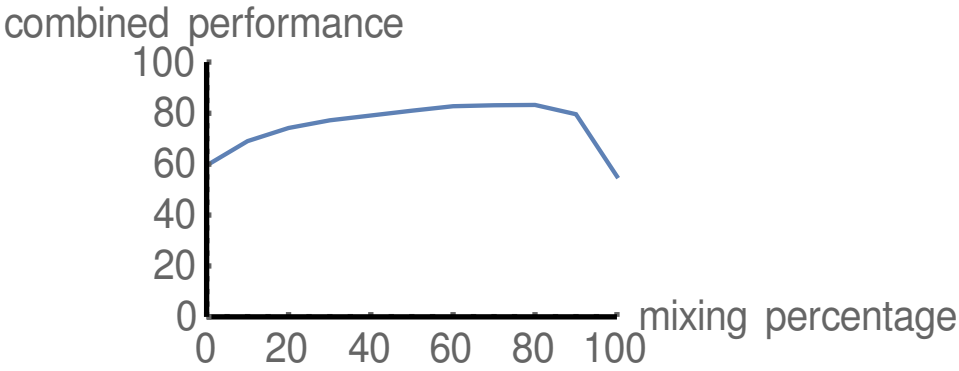


Figure 12. Mixed Training, Top-3 Performance

5.3.3. Iterative Training Data and Analysis

In iterative training, each model is trained with "nice" data first for some number of iterations, then the last two layers are trained with "real" data, for a total of 6000 iterations. The hyperparameters used for all training are those found to be optimal in mixed training: learning rate=0.004 and regularization factor=0.5. The best accuracies and performance of all iterative models trained are shown in Figure 13.

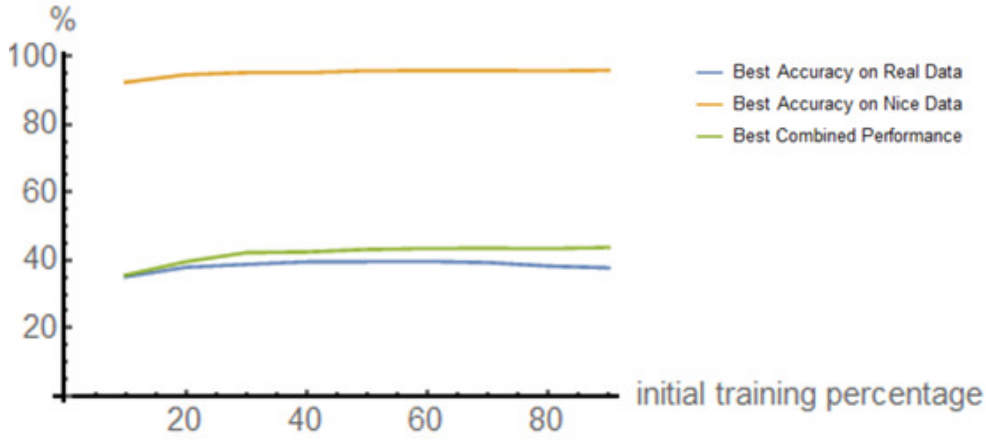


Figure 13. Iterative Training, Top-1 Performance

The results do not support hypothesis 3. Though the best accuracies for real and nice datasets are both consistently average when compared to the mixed training results, the best combined performance is very low, always lower than 45%. Further investigation shows that the restricted training on "real" data never improves the combined performance for any of the 9 models. The core problem is a fast forgetting issue. For example, the various accuracies for the 50% "nice" data iterations model during the "real" data training are shown in Figure 14. Though the "nice" accuracy at iteration 0 is very high (~95%) due to the prior "nice" data training, it quickly drops to 20% within 200 iterations of training with only "real" data. Therefore, as the "real" data accuracy rises, "nice" data accuracy drops, resulting in low combined performance.

From another perspective, iterative training also does not benefit accuracies of individual datasets, as no individual accuracy exceeded the best results from mixed training. This can be explained by the limited flexibility due to the restrictiveness of only allowing training of the last two layers.

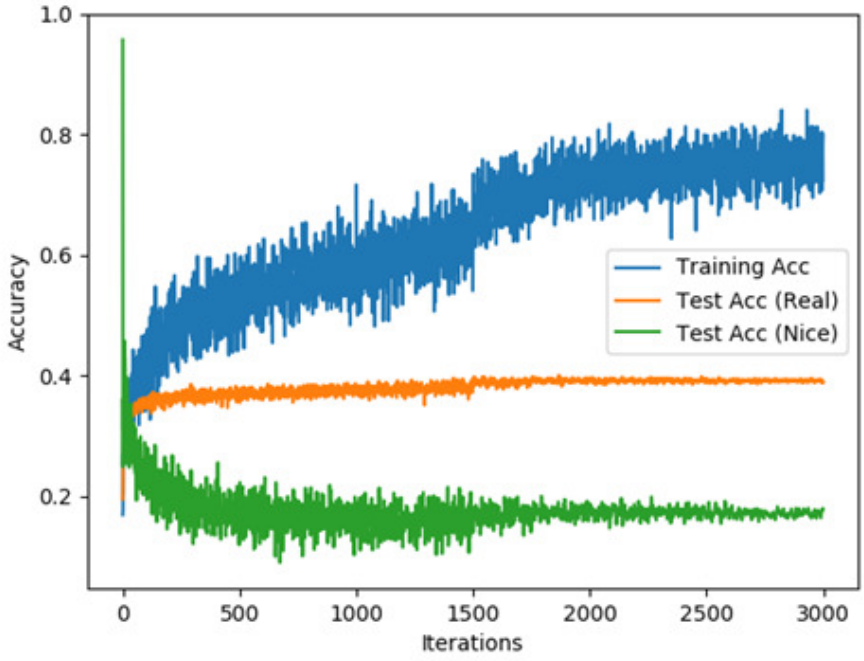


Figure 14. Iterative Training, Accuracies of 50% Model during Restricted Training

6. APPLICATIONS

The method developed in this research can be applied to scenarios of image classification in which two datasets with the same categories, one more difficult to learn than the other, are available, and good accuracy is desired on both datasets using a single neural network model. This allows for a wide variety of applications, for example:

- An autonomous supermarket, where both item sorting (cleaner images) and customer checkout (noisier images) can be done with image classification, and both types of images are widely available online.
- Types of object identification (e.g. species of flowers, trees, or minerals), where both finely preserved samples (cleaner images) and fieldwork images (noisier images) are available, and clients may use the network to identify both types of images.
- Product searching with an image of the product, which is a subcategory of object identification. For example, an application can apply the methods presented in this paper to identify the brand and make of a coat in an image, and then redirect a user to a seller of that coat.
- Online learning with a similar scenario is possible if there are two characteristically different data sources which provide consistently clean/noisy data over time. In such a scenario, buffer the incoming images from both data sources, and then train with batches mixing images from the buffer at the aforementioned ratio of 3 : 7.

7. LIMITATIONS

The limitations of the results of this research are listed here:

- The study is constrained to convolutional neural networks, specifically the ResNet model. Other machine learning approaches, such as genetic algorithms and other types of neural networks, may behave differently. However, CNNs that are similar to ResNet are expected to

exhibit similar behavior. Similarly, the study is constrained to image classification tasks and differences may be observed for other tasks.

- The images used as dataset in this study are relatively small in size (all images are normalized to 100x100 px). This also implies a thin network, with few trainable parameters every layer compared to CNNs used in most other studies. A larger network may produce different results.
- Both training methods studied use only one trained CNN to perform the classification task. Methods that use more than one model may perform better in accuracy, but would be more computationally expensive.

8. CONCLUSION

Two different methods of training with two characteristically different datasets with identical categories were proposed and studied. The model used for the study was a neural network derived from ResNet. The correlation between the two datasets caused a higher-than-random accuracy on one dataset when only data from the other dataset was used to train the model. Mixed training was shown to produce the best accuracies for each dataset when the dataset is mixed into the training set at the highest proportion, and the best combined performance when slightly more of the more difficult to learn dataset was mixed in than the other dataset. Iterative training was shown to have a worse combined performance than mixed training due to the issue of fast forgetting, and also performs worse than mixed training in terms of accuracies of individual datasets.

ACKNOWLEDGEMENTS

The author would like to thank the Chinese Academy of Sciences for providing computational power for this research.

REFERENCES

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). Deep residual learning for image recognition. Retrieved from Arxiv database. (Accession No. arXiv:1512.03385v1)

[2] Muresan, H., & Oltean, M. (2018). Fruit recognition from images using deep learning. Acta Universitatis Sapientiae, Informatica, 10(1), 26-42. Abadi, M., Agarwhal, A., Barham,P., Brevdo, E., Chen, Z., Citro, C., . . . Zheng, X. (2016, March 16).

[3] TensorFlow: Large-Scale machine learning on heterogeneous distributed systems. Retrieved from Arxiv database. (Accession No. arXiv:1603.04467v2)

AUTHOR

Yuxuan BaoYuxuan is a high school senior at Northfield Mount Hermon School in Gill, MA. He enjoys compu ter science very much and is excited about the capabilities of artificial intelligence. He is in the platinum division of the USACO contest.