Soft Voting System for Image Classification on CIFAR-10: Optimization Approach

Pedram Pasandide

Acknowledgment

This is a course project in CSE790 at McMaster University presented by Prof Ben Bolker. I am thankful for their invaluable guidance and supervision throughout this project.

Introduction

Image classification is a computer vision technique in which an algorithm is trained to identify objects or features in an image. It is one of the core areas of computer vision and has numerous applications, including object recognition, scene analysis, and image retrieval. The goal of image classification is to assign a class label to an image based on its content.

Over the years, image classification has evolved from traditional hand-crafted features and shallow models to deep learning-based approaches that automatically learn complex features from large amounts of data. Some of the most significant works in this area include AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan et al., 2014), GoogLeNet (Szegedy et al., 2014), ResNet (He et al., 2015), and DenseNet (Huang et al., 2016).

AlexNet, developed by Alex Krizhevsky et al., was the first deep neural network to achieve significant accuracy on the ImageNet dataset, a large collection of images used for training and evaluating computer vision models. AlexNet used convolutional neural networks (CNNs) and introduced techniques such as data augmentation and dropout regularization to reduce overfitting.

VGGNet, developed by the Visual Geometry Group at the University of Oxford, was designed to improve on AlexNet by increasing the depth of the network and using smaller filters in the convolutional layers. VGGNet demonstrated that deep networks could be trained effectively with a simple design and outperformed AlexNet on the ImageNet dataset.

GoogLeNet, developed by Google, introduced the concept of Inception modules, which allowed the network to learn multiple scales of features simultaneously. GoogLeNet achieved state-of-the-art performance on the ImageNet dataset at the time and was one of the first networks to use multiple branches and parallel convolutional layers to increase the capacity of the network.

ResNet, developed by Microsoft Research, introduced the concept of residual connections, which allowed the network to learn even deeper networks with hundreds of layers. ResNet showed that deep networks can be trained effectively and achieve high accuracy on complex datasets like ImageNet.

DenseNet, developed by Gao Huang et al., introduced the idea of dense connections, which allowed the network to propagate information between all layers of the network. DenseNet demonstrated that dense connections can improve the accuracy of deep networks and reduce the number of parameters in the network. The Table 1 shows a summary of the mentioned models

Image classification has come a long way since its early days and continues to evolve as new models and techniques are developed. These works have not only improved the accuracy of image classification algorithms but also paved the way for new applications of computer vision.

Table 1: A comparison between the models

Model	Neurons/Hidden Layers	Parameters	Classes	Image Resolution	Accuracy
AlexNet	8 layers (5 convolutional and 3 fully connected)	60 million	1000	256 x 256	Top-1: 56.2% Top-5: 79.2%
VGGNet	19 layers (16 convolutional and 3 fully connected)	138 million	1000	224 x 224	Top-1: 71.5% Top-5: 90.1%
GoogLeNet	22 layers (inception module)	6.8 million	1000	224 x 224	Top-1: 69.8% Top-5: 89.6%
ResNet	152 layers	60 million	1000	224 x 224	Top-1: 76.3% Top-5: 93.3%
DenseNet	121 layers	7 million	1000	224 x 224	Top-1: 74.9% Top-5: 91.7%

Methodology

Voting system is a method of combining the predictions of multiple models in machine learning to make a final prediction, in which two schemes are common among classifier, including hard voting and soft voting (Kelleher et al., 2015).

Hard voting involves combining the predictions of multiple models by taking a majority vote. The final prediction is based on the class label that has the highest number of votes from the individual models (Kelleher et al., 2015). This method is simple and straightforward but can be affected by bias if one model is consistently more accurate than others (Rokach, 2010).

Soft voting, on the other hand, involves combining the predicted probabilities of each class from multiple models and making the final prediction based on the class with the highest average probability (Kelleher et al., 2015). This method is more nuanced and takes into account the level of confidence each model has in its prediction (Rokach, 2010). Soft voting can result in better performance compared to hard voting, especially when the models have a high degree of correlation (Kelleher et al., 2015).

The main advantage of hard voting is that it is simple to implement and computationally efficient (Kelleher et al., 2015). It is also less prone to overfitting compared to soft voting (Rokach, 2010). The main drawback of hard voting is that it can be affected by bias if one model is consistently more accurate than others (Rokach, 2010).

The main advantage of soft voting is that it takes into account the level of confidence each model has in its prediction, resulting in improved performance compared to hard voting (Kelleher et al., 2015). The main drawback of soft voting is that it can be computationally expensive and more prone to overfitting compared to hard voting (Rokach, 2010).

The mathematical formulation of soft voting in machine learning is based on the idea of combining the predictions of multiple models in order to make a final prediction. Let's assume m models have been developed, with predicted probabilities $p_1, p_2, \dots, p_m$ for a particular sample. The weighted average of the predicted probabilities is given by:

$$p = \frac{\sum_{i=1}^{m} w_i \times p_i}{\sum_{i=1}^{m} w_i}$$

Where w_i is the weight for the prediction of each model, and p is the predicted class with the highest probability. If the optimization algorithm is able to handle linear constraints, weights can be normalized between 0 to 1 with the following constraint:

$$\sum_{i=1}^{m} w_i = 1$$

The problem with soft voting is that each neural network for classification, like CNN in the last layer, a function like max function is applied to choose the highest probability of each class. This means a hard voting is already applied to the probability vector.

To tackle this problem, it is more efficient to apply soft voting on the probability vector of each model instead of final predictions.

Let's say we have 10 classes and 3 neural networks with different algorithms. $y_c^{n,f}$ is the final prediction of each of the model n where n=1,2,3 is the counter for the models, and c=c1,…,c10 is the predicted class by each model n. $y_c^{n,f}$ is computed by the following formula:

$$y_{ck}^{1,f} = max[y_{c1}^{1,LL}, y_{c2}^{1,LL}, y_{c3}^{1,LL}, y_{c4}^{1,LL}, y_{c5}^{1,LL}, y_{c6}^{1,LL}, y_{c7}^{1,LL}, y_{c8}^{1,LL}, y_{c9}^{1,LL}, y_{c10}^{1,LL}]$$

Where $y_{ci}^{1,LL}$ is the probability of class ci in the last layer (LL) predicted by first model (n=1). $y_{ck}^{1,f}$ is equal to the higher probability. For example, if:

$$y_{c5}^{1,LL} > y_{c1}^{1,LL}, y_{c2}^{1,LL}, y_{c3}^{1,LL}, y_{c4}^{1,LL}, y_{c6}^{1,LL}, y_{c7}^{1,LL}, y_{c8}^{1,LL}, y_{c9}^{1,LL}, y_{c10}^{1,LL}$$

so

$$y_{ck}^{1,f} = y_{c5}^{1,LL}$$

It means class c5 is the predicted class by first model. The same procedure is happening in other two models:

$$y_{ck}^{2,f} = max[y_{c1}^{2,LL}, y_{c2}^{2,LL}, y_{c3}^{2,LL}, y_{c4}^{2,LL}, y_{c5}^{2,LL}, y_{c6}^{2,LL}, y_{c7}^{2,LL}, y_{c8}^{2,LL}, y_{c9}^{2,LL}, y_{c10}^{2,LL}]$$

$$y_{ck}^{3,f} = max[y_{c1}^{3,LL}, y_{c2}^{3,LL}, y_{c3}^{3,LL}, y_{c4}^{3,LL}, y_{c5}^{3,LL}, y_{c6}^{3,LL}, y_{c7}^{3,LL}, y_{c8}^{3,LL}, y_{c9}^{3,LL}, y_{c10}^{3,LL}]$$

For example:

$$y_{ck}^{2,f} = y_{c9}^{1,LL} \text{ and } y_{ck}^{3,f} = y_{c5}^{3,LL}$$

Following equation describes the conventional method for soft voting.

$$y_{ck}^{V,f} = max[\omega_1 \times y_{ck}^{1,f} \ , \omega_2 \times y_{ck}^{2,f} \ , \omega_3 \times y_{ck}^{3,f}]$$

Where $y_{ck}^{V,f}$ is the final prediction by the voting system. Of course we find ω_i with optimizations. **But we already missed some information** since in $y_{ck}^{1,f}$ *and* $y_{ck}^{2,f}$ *and* $y_{ck}^{3,f}$ are obtained by a max function, especially if Top-2 accuracy is higher than the Top-1 accuracy. (NOTE: If $\omega_1 = \omega_2 = \omega_3$ it is hard voting, and based on our example $y_{ck}^{Final} = y_{c5}^{3,LL}$ since it was predicted two times).

To solve the problem, we can formulate the soft voting system using the following formulation:

Model 1 weighted		Model 2 weighted		Model 3 weighted		Summation
$\omega_1 \times y_{c1}^{1,LL}$	$+$	$\omega_2 \times y_{c1}^{2,LL}$	$+$	$\omega_3 \times y_{c1}^{3,LL}$	$=$	y_{c1}^{V}
$\omega_1 \times y_{c2}^{1,LL}$	$+$	$\omega_2 \times y_{c2}^{2,LL}$	$+$	$\omega_3 \times y_{c2}^{3,LL}$	$=$	y_{c2}^{V}
$\omega_1 \times y_{c3}^{1,LL}$	$+$	$\omega_2 \times y_{c3}^{2,LL}$	$+$	$\omega_3 \times y_{c3}^{3,LL}$	$=$	y_{c3}^{V}
$\omega_1 \times y_{c4}^{1,LL}$	$+$	$\omega_2 \times y_{c4}^{2,LL}$	$+$	$\omega_3 \times y_{c4}^{3,LL}$	$=$	y_{c4}^{V}
$\omega_1 \times y_{c5}^{1,LL}$	$+$	$\omega_2 \times y_{c5}^{2,LL}$	$+$	$\omega_3 \times y_{c5}^{3,LL}$	$=$	y_{c5}^{V}
$\omega_1 \times y_{c6}^{1,LL}$	$+$	$\omega_2 \times y_{c6}^{2,LL}$	$+$	$\omega_3 \times y_{c6}^{3,LL}$	$=$	y_{c6}^{V}
$\omega_1 \times y_{c7}^{1,LL}$	$+$	$\omega_2 \times y_{c7}^{2,LL}$	$+$	$\omega_3 \times y_{c7}^{3,LL}$	$=$	y_{c7}^{V}
$\omega_1 \times y_{c8}^{1,LL}$	$+$	$\omega_2 \times y_{c8}^{2,LL}$	$+$	$\omega_3 \times y_{c8}^{3,LL}$	$=$	y_{c8}^{V}
$\omega_1 \times y_{c9}^{1,LL}$	$+$	$\omega_2 \times y_{c9}^{2,LL}$	$+$	$\omega_3 \times y_{c9}^{3,LL}$	$=$	y_{c9}^{V}
$\omega_1 \times y_{c10}^{1,LL}$	$+$	$\omega_2 \times y_{c10}^{2,LL}$	$+$	$\omega_3 \times y_{c10}^{3,LL}$	$=$	y_{c10}^{V}

Eventually, the maximum of new probability vector will be considered as the prediction of the soft voting model.

$$y_{ck}^{V,f} = max[y_{c1}^{V} \ , y_{c2}^{V} \ , y_{c3}^{V} \ , y_{c4}^{V} \ , y_{c5}^{V} \ , y_{c6}^{V} \ , y_{c7}^{V} \ , y_{c8}^{V} \ , y_{c9}^{V} \ , y_{c10}^{V}]$$

The aim of this project is to find the optimal weights of soft voting ($w_1, w_2, and\ w_3$), using a gird search algorithm to reach the highest accuracy on validation dataset.

CNN (Convolutional Neural Network), and ResNet50, and ResNet50 are the methods which is used to classify the images on CIFAR-10 dataset.

Grid Search

The choice of using grid search versus other optimization techniques depends on the problem at hand and the resources available. Grid search is a simple and straightforward method for hyperparameter tuning and is widely used in practice. However, it may not be the most efficient method for high-dimensional hyperparameter spaces or if the number of samples is large.

Other optimization techniques such as random search, Bayesian optimization, or gradient-based methods may be more efficient in these cases. It's important to note that no single method is universally better, and the choice of optimization technique should be based on the specific problem and available resources. In the case of the soft voting weights for multiple models, grid search may be a reasonable choice if the hyperparameter space is not too large.

Grid search can be computationally costly, especially if the step size is low. On the other hand, a lower step size with higher precision can be significantly helpful to achieve higher accuracy. In this project, three weights are chosen between 0 to 1 with step size 0.1. A grid search is applied in a loop to find the highest accuracy. To make sure there would be a RAM limitation, intermediate results are deleted, and only the best model will be kept.

Results

All models were trained on GPU to compute faster. I have used Google Colab pro with GPU NVIDIA A100-SXM4-40GB. The CIFAR-10 dataset consists of 60000 RGB images with the size of 32x32 and 3 channels and 10 classes, with 6000 images from each class. There are 10000 test images and 50000 training images. The dataset is already divided into one test batch and five training batches. The test batch contains exactly 1000 randomly selected photos from each class. Since the range of values are from 0 to 255, all the images were divided to 255 with float64 to keep the accuracy as much as possible. This part can be helpful for pre-train models, like ResNet34 or ResNet50, which have been train on dataset with range from 0 to 1.

CNN model

Even a simple CNN model with only seven layers could I achieve to an accuracy of 68% on test data (CNN_model1). In CNN_model2 more layers were added and the accuracy for train, validation, and test achieve to 82%, 81%, and 80%, respectively. Since there was no sign of overfitting, only 10% of original train data set was randomly selected for validation.

Besides adding more layers to CNN_model3 compared with the previous CNN model, a global optimizer, SGD, was used instead of Adam. The learning rate was initially set to 0.001 and an early stopping criteria was set to avoid overfitting. A learning rate schedule was set with a factor of 0.1 and monitoring validation loss and patient equal to 20. The model stopped on epoch equal to 215, with test accuracy over 75% while learning rate at this point was 0.0001. To make sure that this is the highest accuracy, the learning rate was set back to 0.001 and the model was train for more epochs.

Since there was not significant increase and there was no sign of overfitting (a close accuracy for all train, validation, and test dataset), a fine-tune was performed by changing optimizer to a local optimize, in this case Adam, and slight changes on learning rate schedule and early stopping to avoid overfitting. An early stopping happened on epoch 29 while the learning rate was dropped to 0.0001. Figure 1 and Figure 2 shows the accuracy and loss function, respectively. The sudden change around epoch 320 is due to the change of optimization algorithm to Adam and initialing the learning rate with a lower value. Finally the accuracy of this model could achieve to over 87% (a more detailed results in Table 2).

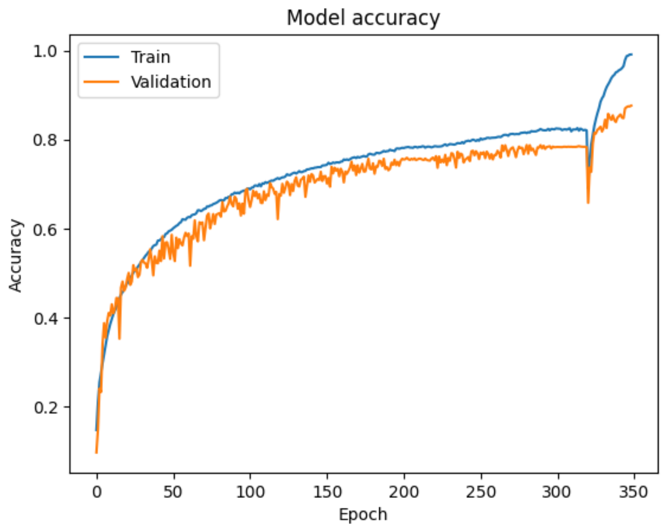

Figure 1: accuracy over epochs for CNN_model3 (the best CNN model)

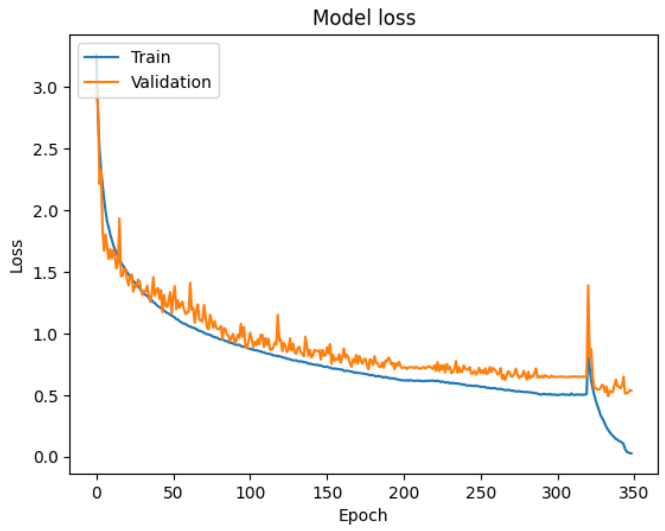

Figure 2: Loss value over epochs for CNN_model3 (the best CNN model)

ResNet-50

Since the trainable parameters in ResNet-50 is high, using a global optimizer can be computationally costly. In all ResNet models, Adam optimizer was used. In a simple model (ResNet50_model1), even though the accuracy of train data reached to 95%, the validation and test accuracy were around 75% indicating an over-fitted model. In ResNet50_model3, data augmentation was performed to avoid over-fitting. Learning rate, which was initialized with 0.001, was dropped to 0.00001 at the end of train with 28 epochs. The accuracy for all training, validation and test data was around 85%. More details are provided on Table 2. Figure 3 and Figure 4 shows the accuracy and loss function of ResNet50_model3.

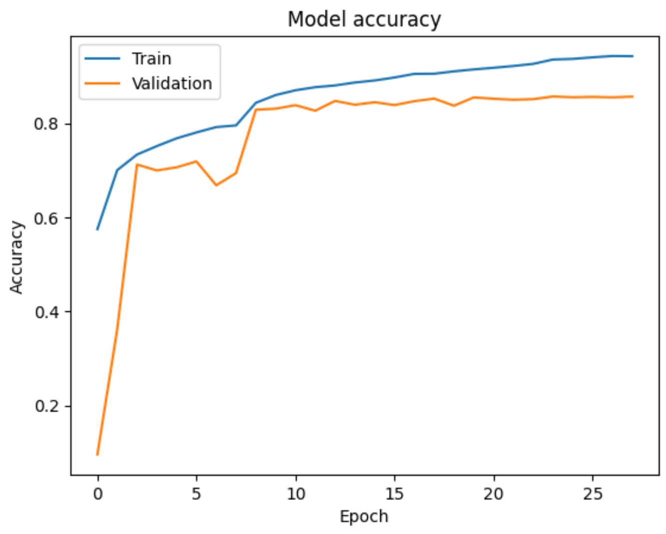

Figure 3: accuracy over epochs for ResNet50_model3 (the best ResNet-50 model)

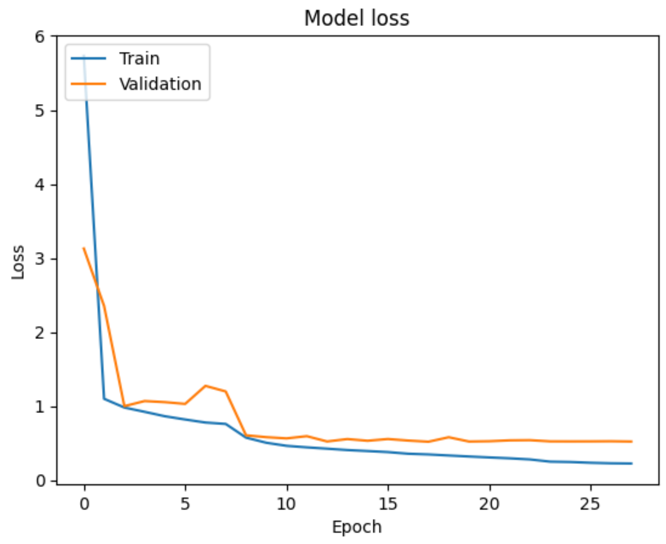

Figure 4: accuracy over epochs for ResNet50_model3 (the best ResNet-50 model)

ResNet-34

Since ReNet34 is no available on Tensorflow, a model from scratch is developed, and all the hyperparameters were tuned only by trying different values. The conv1x1 function applies a 1x1 convolutional layer with batch normalization and ReLU activation function to the input tensor. The purpose of the 1x1 convolutional layer is to reduce the number of channels in the input tensor.

A function conv3x3 applies a 3x3 convolutional layer with batch normalization and ReLU activation function to the input tensor. Batch normalization is used to normalize the activations of the previous layer. Normalizing the activations ensures that the scale of the input to each layer is roughly the same, which can speed up training and improve the performance of the model.

The residual_block function defines a residual block with two convolutional layers. The first convolutional layer applies a 3x3 filter to the input tensor, and the second convolutional layer applies another 3x3 filter to the output of the first layer. The input tensor is also passed through a 1x1 convolutional layer if the stride is not equal to 1 or if the number of channels in the input tensor does not match the number of filters in the residual block. The output of the second convolutional layer is then added to the input tensor to produce the final output of the residual block. The stride parameter determines the step size of the convolutional kernel when applied to the input. A stride of 1 means that the kernel moves one pixel at a time, while a stride of 2 means that the kernel moves two pixels at a time. Using a larger stride value can reduce the spatial dimension of the output feature maps, which can be useful for reducing computation and memory usage in deep neural networks.

The ResNet34 function defines the ResNet-34 architecture using a series of convolutional and pooling layers, followed by residual blocks. The input tensor is first passed through a 7x7 convolutional layer with a stride of 2 to reduce the size of the input tensor. This is followed by a batch normalization and ReLU activation function. The input tensor is then passed through a 3x3 max pooling layer with a stride of 2 to further reduce the size of the input tensor. After this, the input tensor is passed through a series of residual blocks, with different numbers of filters and strides. The output of the final residual block is passed through a global average pooling layer, and then through a dense layer with a softmax activation function to produce the final output of the model.

The learning rate schedule used for ResNet-50 was not helpful here. So a manual learning rate schedule based on the number of epochs is used to drop the learning rate from 0.01 to 0.00001 if the training was not stop due to early stopping criteria. A simple model like resnet34_model1 achieve to accuracy around 100% for train and 80% for both validation and test data indicating an over-fitting.

Next model, resnet34_model2, was developed to avoid over-fitting by data augmentation the same as resnet50_model3. Since the image sizes are 32*32, only minor data augmentations could be helpful. Finally, the learning rate schedule was tuned after a few times training the model. Due to early stopping the training was stopped at epoch 22, with accuracy of 83% for test data.

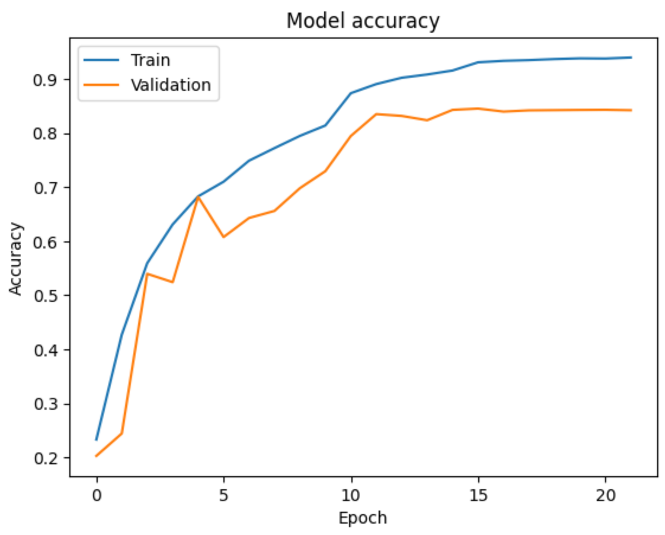

Figure 5: accuracy over epochs for ResNet34_model2 (the best ResNet-34 model)

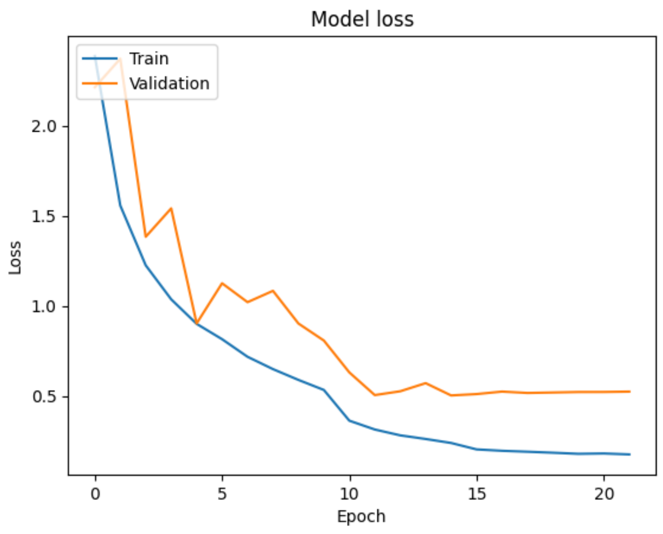

Figure 6: accuracy over epochs for ResNet34_model2 (the best ResNet-34 model)

The Table 2 show a summary of the best models. Two methods were used to find the optimal weights for soft voting between models using validation dataset. First, a grid search was performed and the best accuracy was 92.81% for validation dataset. Second, a Bayesian optimization was conducted and the accuracy was around 92.71% which is close to that of grid search method. Bayesian method outperformed grid search by 1% accuracy over test data. The weights optimized with Bayesian are w1=0.64, w2=0.59, and w3=0.00 for CNN_model3, resnet50_model3, and resnet34_model2, respectively. The results for the weighted_model are shown in the following table.

Table 2: A summary of best models

Model	Size (MB)	Top-1 Acc (%)	Top-2 Acc (%)	Train Acc (%)	Validation Acc (%)	Parameters	Depth
CNN_model3	28.73	87.41	95.33	99.09	87.60	2,493,834	37
resnet50_model3	348.54	85.32	93.89	95.57	85.64	30,412,170	184
resnet34_model2	244.35	83.30	93.24	94.71	84.24	21,309,002	144
weighted_model	377.27	89.03	95.63	99.99	92.71	32,906,007	184

Since the weight for resnet34_model2 is zero, the size of the model is equal to the summation of two other models. The parameter in the weighted_model is equal to the summation of the other two plus 3 for optimized trained. The depth can be considered equal to the maximum depth between two models. Optimization to find the weights with Bayesian is not computationally costly as much as grid search. Although, weighted_model has more parameters, it outperformed CNN_model3 as the best single model for 1.62%. It might be possible to get higher accuracy if ResNet models were able to achieve higher accuracy. In further research, if the computation cost is not a bottleneck, fine-tuning and using global optimizers can help ResNet models to achieve a better performance.

References:

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2014). Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4700-4708).

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: algorithms, workflows, and examples. MIT press.

Rokach, L. (2010). Ensemble-based classifiers. Artificial Intelligence Review, 33(1), 1-39.