Fine-grained image classification of Korean food


Department of Astronomy

2021135034, IM TAEUK

Introduction

There are several types of Kimchi in Korea. We want to classify Kimchi by the types in this paper. Especially for here, we classify Kimchi in 11 different classes with 7700 different images of Kimchi. ('갓김치', '깍두기', '나박김치', '무생채', '배추김치', '백김치', '부추김치', '열무김치', '오이소박이', '총각김치', '파김치')

1. Classification with MLP

1-1. Describe model details

Learning rate: 0.001 Weight decay: 0.001 Batch size : 100 Epochs: 10

Layers : 3 Fully connected layers

1-2. The reason for using these hyperparameters

Learning rate:

Deciding the learning rate was totally up to experiences and trials.

I could find the recommended figures for learning rate – 0.01, 0.005, 0.002, 0.001 – but we can't know the best learning rate until we actually try because it depends on our data set, model details, and so on.

So I've tried 0.01, 0.005, 0.001.

0.01 shows the poorest performance.

0.005 was not as bad as 0.01 but also not as good as 0.001.

0.001 shows the best performance.

Of course, it takes long time to train our model with 0.001, but we could use GPU from google colab, so we could save some time.

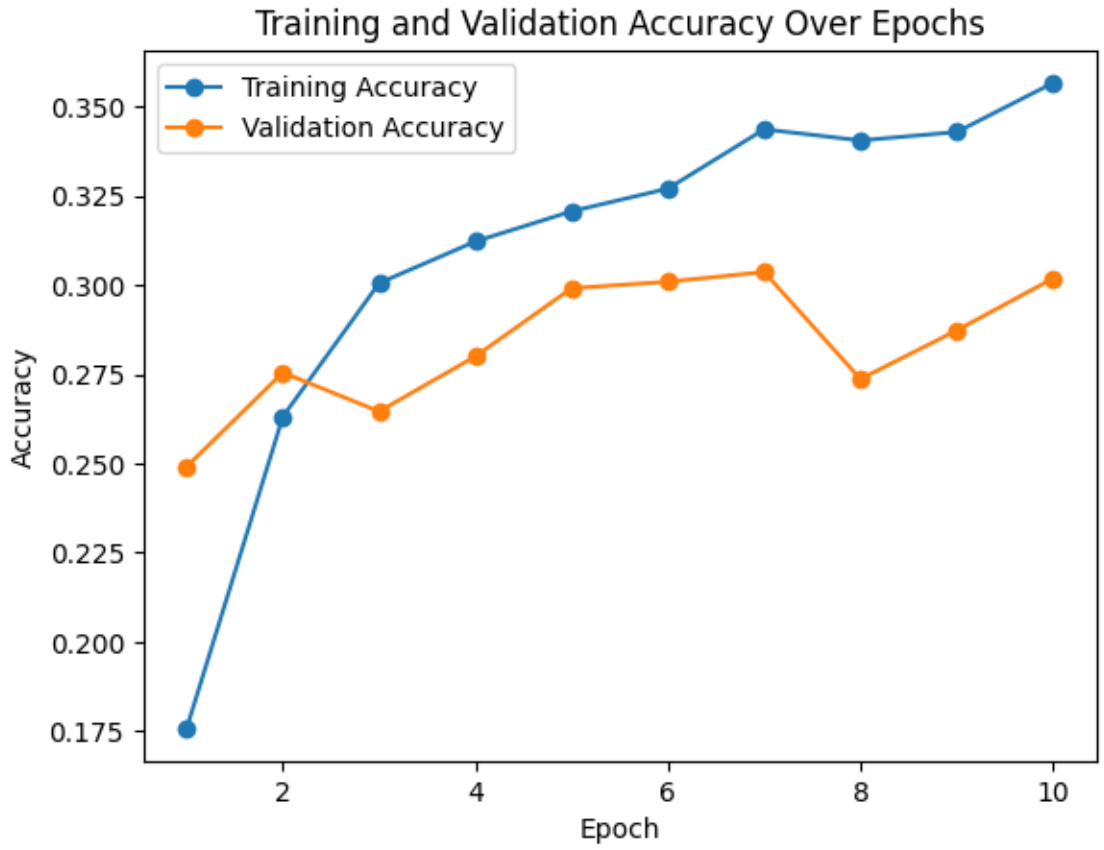
And the biggest problem of low learning rate is the problem of being stuck in

local minima. But since we use the optimizer with stochastic gradient descent, we could assume that as long as the learning rate is not too small, our optimization process will not stuck in the local minima.

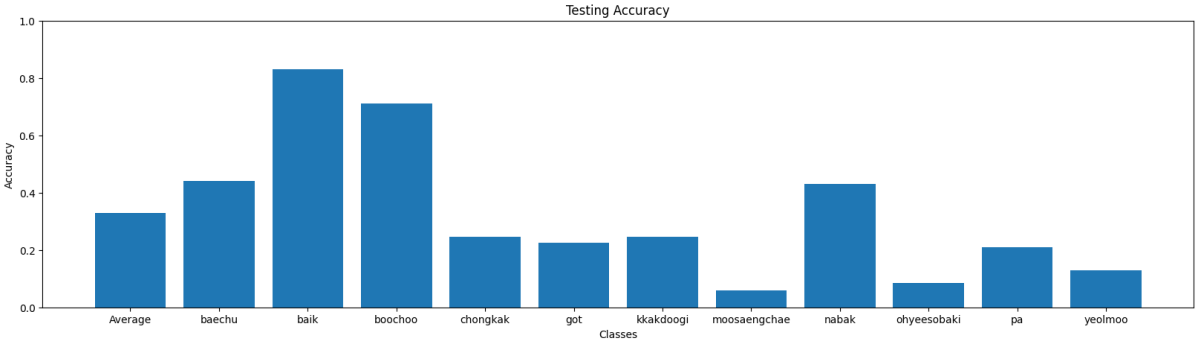
So after numerous trials, we could reasonably think that 0.001 is the best number for learning rate.

Epochs: Since we are using DNN with small data set, many layers didn't contribute to notable improvements of our model. Actually, more epochs could lead to overfitting problem since we don't have many parameters as CNN. So I chose 10 epochs to train our model.

1-3. The training and testing accuracy in a plot



1-4. Class-wise test accuracy in a plot



1-5. The possible reasons for the bad performance in some classes

DNN doesn't capture any feature of image, so we don't know why it is good at some classes and it is not good at other classes.

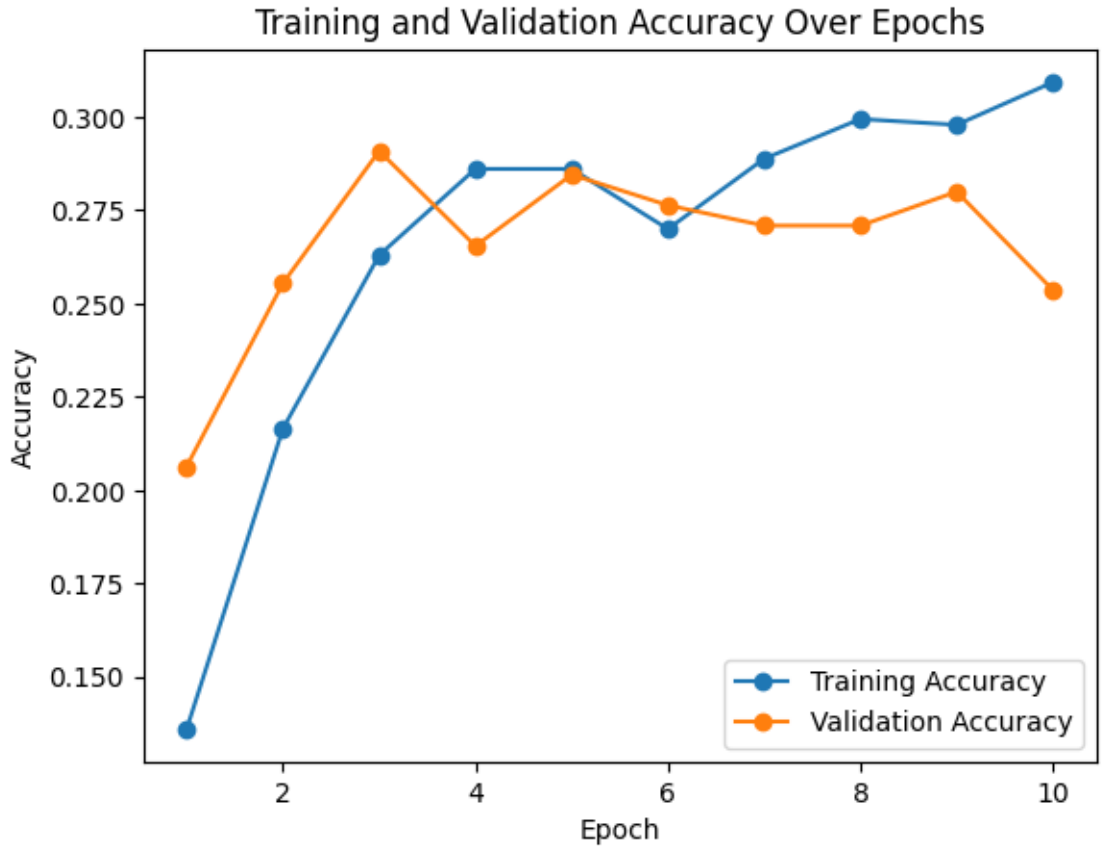
But for baik, we can infer that it's color maybe contribute to its great accuracy. Since other Kimchi have all red or green color while baik only has white color, it maybe the reason why it has good accuracy.

1-6. The ideas to improve the accuracy

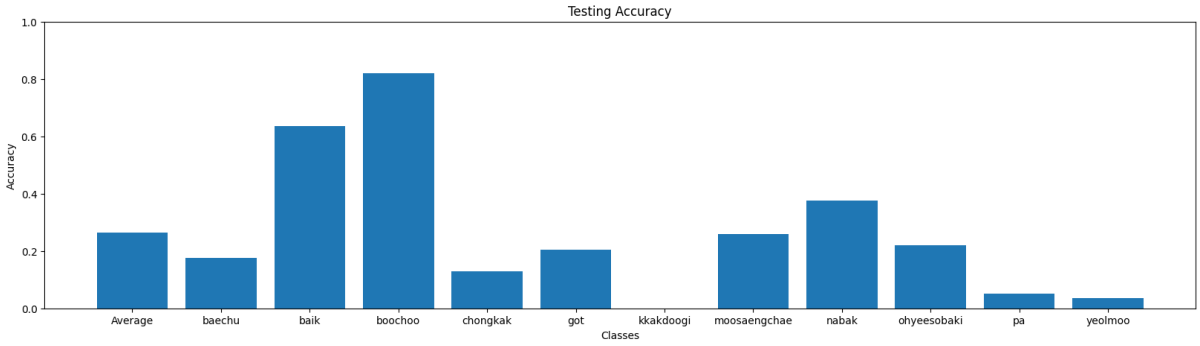
Since we have limited data set, it would be great idea to improve the data. If we could increase the data set, we can also increase the layers. So to improve our performance, we could augment the data set, increase the layers and use some drop out in the layers.

1-7. Apply above idea to improve the accuracy.

I used rotation, normalize, flipping to increase the data set. Also I increased the layers from 3 to 5 and put some drop out among those layers to get the effect of ensemble.



1-8. Plotting class-wise test accuracy and the average test accuracy.



For unknown reasons, we got poorer performance than before with augmented data set. We don't know the reason why it shows the poorer performance. But we can think about a possible reason.

We augmented the data, but there is a possibility that augmented data is not consistent with existing data. So, it only confused the model for training.

2.Classification with deep convolutional neural network(CNN)

2-1. Describe model details

Learning rate: 0.001 Weight decay: 0.001 Batch size : 35 Epochs: 15

Layers : 4 Convolutional layers , 2 Fully connected layers

Dropout : 0.25 for first and third convolutional layers and 0.5 for first fully connected layer.

2-2. The reason for using these hyperparameters

Learning rate:

Deciding the learning rate was totally up to experiences and trials.

I could find the recommended figures for learning rate – 0.01, 0.005, 0.002, 0.001 – but we can't know the best learning rate until we actually try because it depends on our data set, model details, and so on.

So I've tried 0.01, 0.005, 0.001.

0.02 shows the poorest performance whose accuracy was equals to 25%.

I could draw a conclusion that 0.01 is too big to find global minima. Since learning rate was too big, our model just passed some minima and stuck in weird place.

0.005 was not as bad as 0.01 but also not as good as 0.001.

0.001 shows the best performance whose accuracy was nearly 60%.

Of course, it takes long time to train our model with 0.001, but we could use GPU from google colab, so we could save some time.

And the biggest problem of low learning rate is the problem of being stuck in local minima. But since we use the optimizer with stochastic gradient descent, we could assume that as long as the learning rate is not too small, our

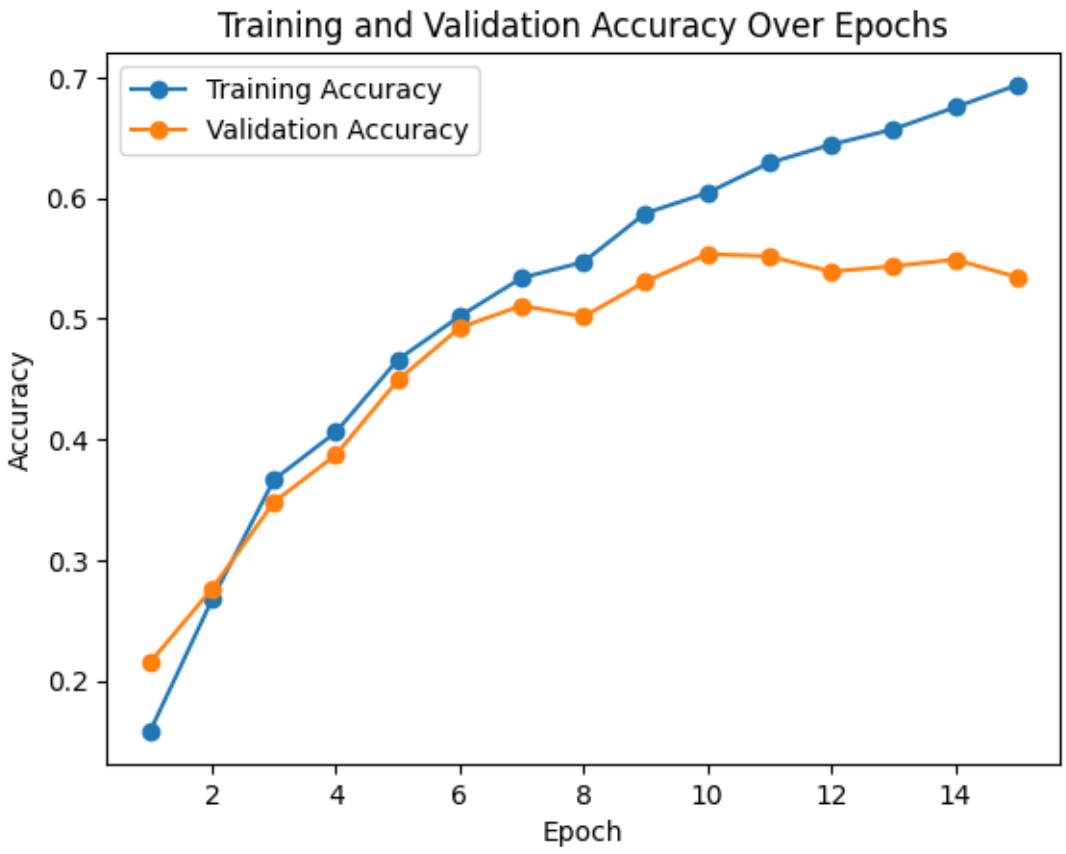
optimization process will not stuck in the local minima.

So after numerous trials, we could reasonably think that 0.001 is the best number for learning rate.

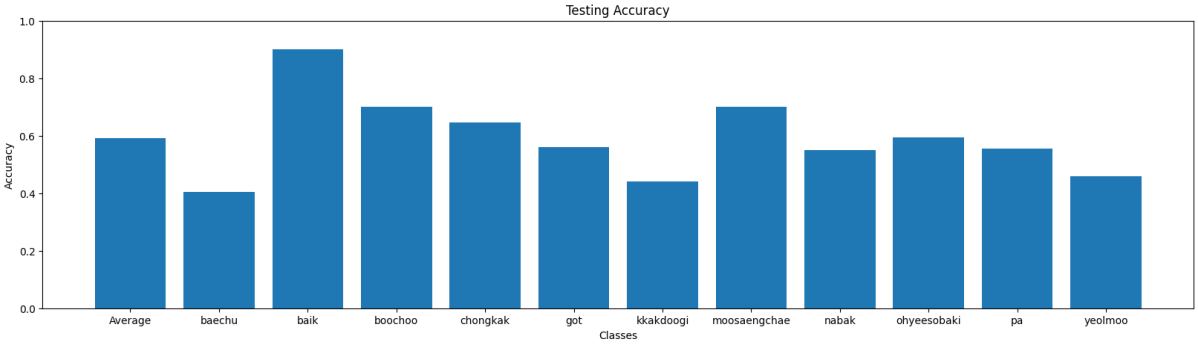
Epochs: If we look at the figure in the next section, we can see that validation doesn't go up well after 9th epochs.

At the first time, I tried 20 epochs. But validation doesn't go up after 9th epochs so I could rationally reason that there will be the risk of overfitting when we train with more epochs. So, after several times of adjusting the number of epochs, I chose it as 15.

2-3. The training and testing accuracy in a plot



2-4. Class-wise test accuracy in a plot



2-5. The possible reasons for the bad performance in some classes

For the performance, usually the model and the number of data is the biggest reason. But since we used the same model for training and the number of data is the same (700 images) for every class, these are not possible for reasonable answers.

So I looked up to images closely and I could draw a conclusion like below.

1. Baik's high performance

All other Kimchi is either green or red. Only Baik has the color of white. It means that it has distinguishable feature in image which makes our model to classify it quite well.

2. Moosaengchae's high performance

If we see Moosaengchae, unlike other Kimchi, it is thin and long. Other Kimchi is kind of clump, which means that they don't have many segmental elements. But Moosaengchae, it is long and thin. So for our model, it would be easy to classify Moosaengchae comparably easier than other class since it is segmented.

3. Others

Based on above two assumptions, we can draw a conclusion that other Kimchi

would be pretty same for classification. Because others are either green or red and they are not segmented well.

4. Bad performance on kkakdoogi

Among Kimchi, kkakdoogi shows the worst performance. It's because they are just bunch of red rectangular. Other Kimchi have tails and heads. It means that if we rotate them little bit, they don't look same.

But for kkakdoogi, they are just red rectangular and they don't have tails or heads. They just look same in every picture. If we look at the pictures, even though they are rotated, they look totally same.

So in conclusion, the bad performance on kkakdoogi is due to their lack of distinguishable features such as color, tails, heads.

These 4 inferences is accordance with our CNN model which catches the notable features to train the model.

2-6. Ideas to improve the accuracy

There are five ways to improve the accuracy.

First, data augmentation will be a good approach. Since we have 770 images per each class, they are not enough to train the model. If we can train with more images, our model will have the better performance.

Here, what we have to consider is how we are going to augment the data set. There are various ways to increase the number of data set such as randomly rotation, changing hue, regularization, flipping and so on. We have to be really be careful to pick among those since if the augmented data doesn't get along with existing data, they are useless.

Second, make our model deeper. This method actually has to go together with first method. If there will be more layers, there will be more parameters, so the performance of our model would be better. But the problem is vanishing gradient, overfitting problem.

Since our model is not that deep to consider vanishing gradient problem, we can ignore the problem. But for overfitting problem, we have to think about it. Our data set is too small for deep layers, if we just increase the number of the layers, there will be overfitting problem. So in order to implement this method, data augmentation should be come first.

Third, drop out method would be good to implement. As we know drop out would work as ensemble. If we randomly drop out our parameters, we can get an effect of combining a few small models together.

Next, we can use the method called stratified K-fold. It is widely used for classification model with small data set. Since our data set is small and we are training our model for classification, it is a good approach. As we set more folds, we can expect the effect of ensemble. So I tried K-fold method with 5 folds.

Last but not least, we can use pre-trained model. In this paper, I refer to pytorch manual from google and made the model by myself. But as we know, if we use pre-trained model, it is more easier to train them and get better result. There are many pretrained model, so it would be good if I could adopt the model from others and adjust it to fit our data.

2-7. Appliance of idea to improve the accuracy

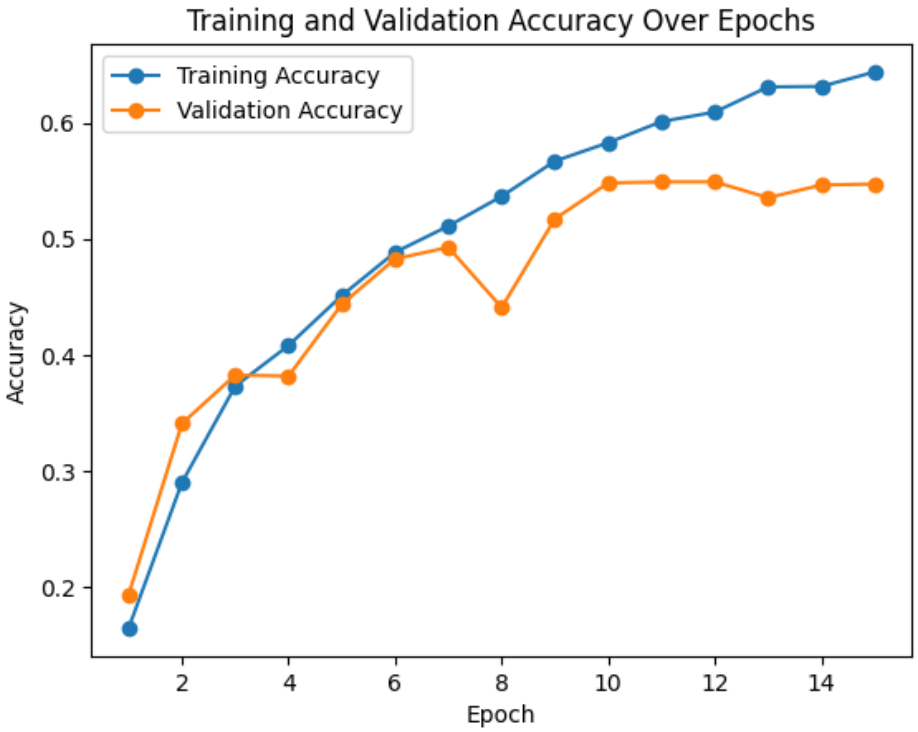
So in this project, I increased one more layer and used drop out for three layers

with augmented data set. To augment the data set, I used rotation, flipping.

At first, I also tried to change the hue and normalize the data set. But for unknown reasons, our model with augmented data doesn't show the better performance. Actually it show the more poor performance.

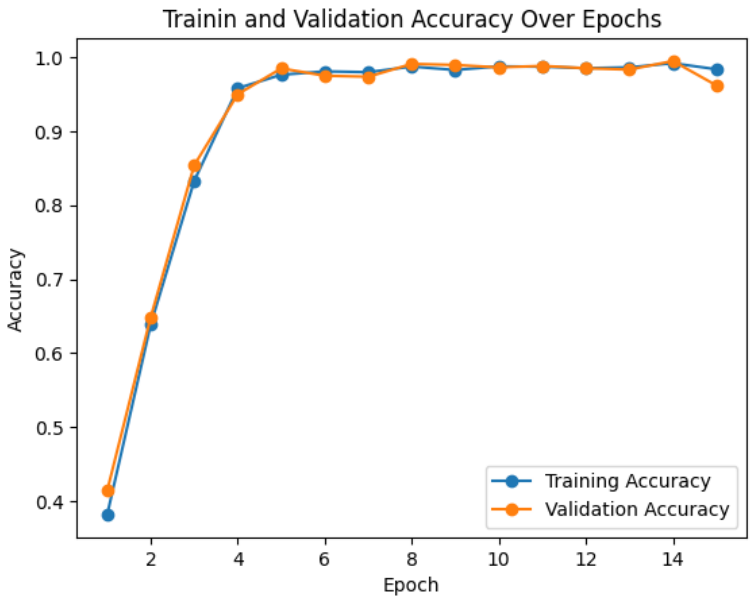
After several times of trials, I found out that data augmentation with just rotation and flipping is the best way to increase the mount of data. Otherwise, maybe augmented data isn't consistent with existing data so it doesn't help for training the model.

Below is the result of our model, after numerous trials of adjusting the parameters.



As we can see, actually the performance of the model with augmented data doesn't show the better accuracy. I think that we should capture the feature of images and then augment the data by classes to get better performance.

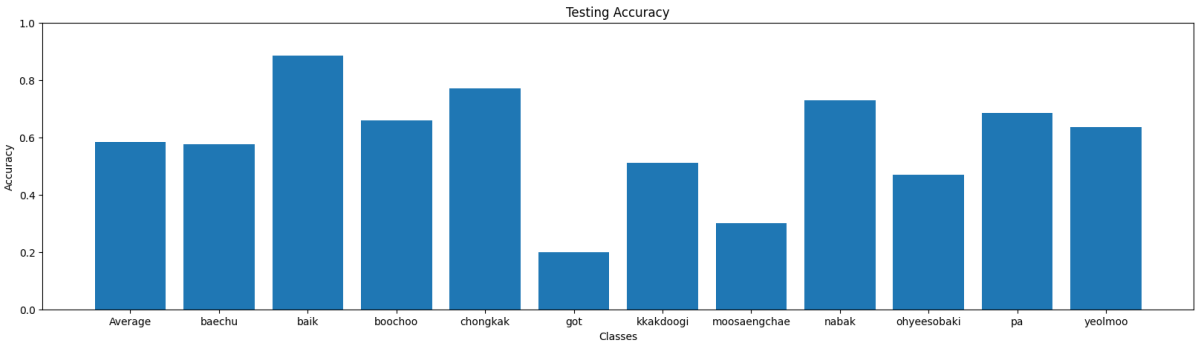
I also implemented the K-fold method.



As we can see, in K-fold, we could get the better training accuracy.

2-8. Class-wise test accuracy and the average test accuracy in a plot

Below is the result of our CNN model with augmented data set. As we can see, this model shows the poorer result in got and moosangchae than before. We increased layer and augmented data set, but this happened.



Below is the result of our CNN model with K fold.

2-9. The difference to the MLP based model.

As we know, CNN captures the feature of image based on geometric characteristics while DNN can’t.

First, CNN doesn’t ignore the color features so called RGB. When we give the input to our model, we actually give with three channels which means the image has the color. But for DNN, we doesn’t give the data to the model with color. We just flattened the image for DNN, which means we ignore the feature.

Second, CNN give some credits for the feature in the image by location, gradient, and so on. Since we give the image in 3 dimension vector to CNN, we don’t ignore the geometric characteristic of the image. And since CNN works with kennels, it can capture some features such as gradient, changing color easily.

So based on above two distinctions between DNN and CNN, we can draw a conclusion like below.

1. CNN takes more time to train than DNN. (more parameters)
2. CNN shows the better performance.
3. CNN is suitable for image classification.

In summary, since CNN can capture the features of image it would be better for image classification even though it takes longer time.

3.Limits

It would be great if we could borrow some pretrained models. Since it was my first time to build an actual model, I couldn’t think of it. Also it was hard to tune hyperparameters. There are some recommendations for batch size, kennel size, folds and so on but there were no answers. So I tried numerous times, but every

time I tried, I could get different results. I failed to get the relation between hyperparameters and the model performance. I think the reason is that there were too many classes and the images in each class have different features.

If we could examine the images one by one, we will get better way to train our model.

Last, I plot the graph about training and testing as training and validation.

We should graph the test accuracy by class-wise in the last section so I thought testing means just validation. Also, we can’t see the test accuracy to tune the hyperparameters, so I thought testing accuracy should be validation accuracy. I found that testing accuracy equals to test accuracy in the last minute, so I couldn’t plot it again. It’s a shame that I couldn’t draw it, but I hope that validation would be fine.