Homework 7 Report

Convolutional Neural Networks – Intel Image Classification

Kaggle Username: pngo1997

Public score: 0.96048

Mai Ngo

DSC 578 Neural Network and Deep Learning - SEC 701

Dr. Noriko Tomuro

November 12, 2023

Table of Contents

Original Model

Same as other homework, I started by breaking down the starter code and input comment/variable name change to my own understanding. For summary, the original model has two convolutional layers, each followed by a respective pooling layer – this to help reduce overfitting and reduces computational cost. Then the output got flatten to a 1-dimensional vector and passed to the two fully connected (FC) dense layers - 128 nodes for first dense layer, and 6 nodes represents six categories for final output layer. Note, the final layer using Softmax activation function. As for building model architecture, I follow your framework guidance from PowerPoint 8 – Advanced Topics on CNNS. First, batch normalization will be applied after activation function layer to stabilize activation values. Second, followed by Pooling layer. Third, followed by Dropout layer.

I actually started by using my final model from Homework 4, the results were not good. After many failed attempts, I was consistently tuning parameters intuitively, yet did not get any 'breakthrough' output. At this point, most of the time train and validation accuracy scores come around 30 – 47%, the model does not perform good, big difference between train and validation scores as well. Then I started looking at the model development guideline from the assignment sheet and decided to follow that. Here are what I experimented and my conclusion for some parameters. Please note that for each parameter, I ran the model 5 or 6 times:

Regularization: I followed my workflow from Homework 4. Since L2 Regularization (Ridge) suppresses features (pixels) of each image as much as it can but not to be exactly zero, this gives reasonable weight distribution and more appropriate in image classification task. I tried L2 penalty on the layer's kernel/bias/output - both individually and combined - of 0.01 to Convolutional, Dense, then to both layers respectively. The output looks really bad and eventually I decided to keep regularization out of the model.

Mini-batch size and learning rate: These are the only 2 parameters that I can see distinguish output given small changes. I did at some point change to mini-batch size of 16, the results were consistent with size of 32. Thus, I kept its original value. As for learning rate, I started with 0.3 then notice the smaller learning rate value, the better the output. Eventually, my fixed learning rate value is 0.0001.

After this, I was trying so many combinations of hyper parameters and layers. Overall, I was running about 60 models, and the outputs got worsen over time.

Non-Best Model

Looking at my Kaggle submission chart, I have so many 'Non-Best' models. And those submission were just a small portion of models I deemed as 'failed'. I tried with so many combinations of parameters and layers, majority of them actually yield great training performance but when it comes to testing, it does not look great.

In addition, I also tried data augmentation. Looking at my **HW7-CNN Image Classification.ipynb** notebook, I had two attempts in applying data augmentation. The first attempt was using ImageDataGenerator, which I initially thought it was a 'break through' moment in this assignment. Specifically using 'flow_from_directory' that allows augmenting data from the get-go for train and validation sets with splits ratio. But the more I train the models, I notice the output does not look right. Despites checking and troubleshooting this set up, I cannot get the desired output.

Eventually, I found out that all the models using ImageDataGenerator augmented data yields incorrect output. It was [1., 0. , 0., 0., 0., 0.] instead of i.e., [0.01480513, 0.9240395 , 0.00640454, 0.00159798, 0.00859121, 0.0445617] – very precise output which it should not be like that. Therefore, for this question I would conclude **all the Non-Best models that I have are those using ImageDataGenerator augmented data.** I also notice models trained from ImageDataGenerator have a unique 'noticeable' difference between train and validation sets. Most of the time, my models performance on both sets are pretty consistent with over/under 5% difference at the last epoch. However, the differences and changes of epoch performance from a model using ImageDataGenerator augmentation are shown below – just one of the examples. This performance patterns are consistent as well, given train accuracy/loss progress smoothly, but validation accuracy/loss jump all over the place. It just took me a while to notice this problem and made change.

	Train Accuracy	Validation Accuracy
Epoch 1	30%	14%
Epoch 2	50%	53%
Epoch 3	55%	22%
Epoch 4	62%	30%
Epoch 5	70%	50%

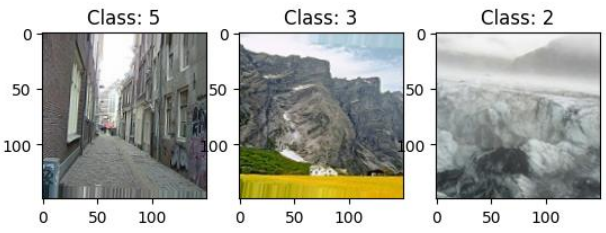


Figure presents train and test set performance on ImageDataGenerator augmented data.

Best Model

Given failure in ImageDataGenerator, I tried a different data augmentation method using Keras 'RandomFlip' and 'RandomRotation', which allows me to apply augmentation directly on the two original train and validation sets. This is when I got an 'actual' breakthrough on this experiment. The augmented data actually works and can properly be used to train my model.

Figure presents Keras 'RandomFlip' and 'RandomRotation' augmented data.

Now, I am going back to train the model from square one. At this point, I have a current-best model saved which lesson learned from overwriting my 'best' score 0.96 model. I continue to tune different combination of parameters until I got a desired output, keep submit csv outputs to Kaggle. Here are my experiences in tuning these parameters:

Number of Convolution layers: I initially started with five set of convolutional layers (CL). Each CL followed with Batch Normalization and a Max Pooling layer size (2,2). I also attempted to apply Average Pooling, but the results were very inconsistent. Eventually, I decided to stay with three set of CLs.

Size of filters: I attempt to use filter size 5*5 and 7*7. The performances with filter size 7*7 somewhat drop noticeably (10%) compared to using filter size of 3*3 and 5*5.

Number of filters for Convolution layers: This is the most interesting parameters to me, because no matter how much I try to navigate this, I could not find a benchmark of how much number of filters to use. I did try both ways, common approach which starts with the smallest filter numbers (32, 64, 128, 256, and 512); and I also tried to the opposite way which begins which largest, knowing that larger number of filters will be able to capture more detailed features. Overall, I could not tell the output differences and decided to stick with filter numbers 32, 64, and 128.

Number and size of Fully Connected (FC) layers/Dropout layers and the percentages: Same as CL, I do not have a benchmark on how many layers to use. I do expect the FC layer size should be reduced up to the final output layer. Eventually, for my best model I use one layer with Dropout implementation of 0.25.

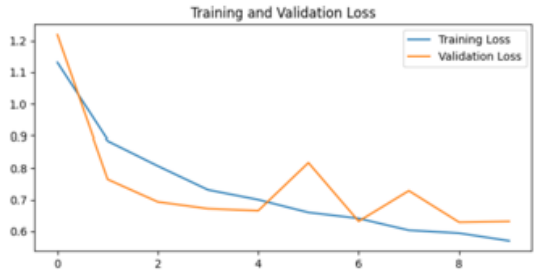
Activation functions, padding = 'same' and strides = 2: I also apply these parameters but cannot see significant improvement. Tried 'selu', 'swish', and 'sigmoid' activation functions.

Below is my best model which yield ranking score of 1.04. I want to emphasize that I also have been trying with more complicated models, i.e., 7 CV, 7 FC layers, with regularizations, etc. At the end, I decided to stick and experiment more with simpler models since it gives me better ranking score and classification results. Also, easier for me to navigate and tune parameters without getting frustrated in this process.

Model 1 - Using augmented data. --> BEST MODEL.

```
model1 = keras.Sequential()
model1.add(keras.layers.Rescaling(1./255, input_shape=(imgHeight, imgWidth,3)))

#CV1 with 32 filters, each of size of 5x5. ReLU activation function.
model1.add(keras.layers.Conv2D(32, (5,5), activation='relu'))
model1.add(keras.layers.BatchNormalization())
model1.add(keras.layers.MaxPooling2D(4,4))

#CV2
model1.add(keras.layers.Conv2D(64, (5,5), activation='relu'))
model1.add(keras.layers.BatchNormalization())

#CV3
model1.add(keras.layers.Conv2D(128, (5,5), activation='relu'))
model1.add(keras.layers.MaxPooling2D(4,4))

#Flattens the output into a one-dimensional array.
model1.add(keras.layers.Flatten())
model1.add(keras.layers.Dense(128, activation='relu'))
model1.add(keras.layers.Dropout(0.25))
model1.add(keras.layers.Dense(6, activation='softmax'))

#Compile model
opt = Adam(learning_rate = 0.0001)  #Set learning rate
model1.compile(optimizer=opt, loss='categorical_crossentropy',
               metrics=['accuracy'])
```

The best model gives me 79.57% of accuracy score on train set, 78.83% on test set. Having 3 Convolutional layers with number of filters of 32 and 64, and 128 and size 5*5. I applied Batch Normalization for the first two CL for training stability and faster convergence – I also tried with different parameters within BatchNormalization() but the results got worst, Max Pooling for the first and last layers. As for Dense layer, I only have one with size 128 and implemented Dropout with 0.25 percentage to avoid overfitting. Overall, this model is very simple, yet the journey took me to this best model was not easy at all. I truly believe I have tried every way possible to improve the model performance at my best capabilities and knowledge.



Reflection

Unlike Homework 4, this homework was difficult for me to keep track of my models and implementation outputs due to plenty number of parameters and combinations. I was all over the place at first trying to get a good grasp of what was going on, how can I choose a good benchmark so I can keep progressively building the model. Basically, trying very hard to gain a good intuition of training this dataset. For my experience in doing this homework, changing one parameter could lead to drastically change in output; and when I change more than two parameters, it gets complicated very fast. I had three stages doing this homework:

- ✍ Understand the original model and start tuning baby parameters one by one.
- ✍ Make the model gradually complex and constantly ended up with bad results.
- ✍ Learn and apply data augmentation (in two ways), then re-training models from square one.

Eventually, I feel like no matter how much progress I thought I was making; I could not improve the model. If you look at the leader board, I got one best score of 0.96 which I ended up overwrite that model's code as well. On top of that, given the graphs represent train and validation scores do not show signs of over or underfitting. Yet, my submission score still not improve. Overall, I am satisfied with the model locally, stick to my usual objective when I build learning models up till now. This homework had given me a different perspective: Good loss and accuracy scores of train and validation sets does not mean the model will perform well on unseen data.

Nonetheless, I still want to use this Homework to learn Keras and TensorFlow, rather than to keep aiming for a good performance model, i.e., passing certain ranking or score threshold. I learned how to implement data augmentation, visualize, and apply it in the models. Thus, this was a good learning curve for me. One final extra knowledge is prefetching both train and validation sets to boost execution performance and maximize GPU memory utilization.

For me, the 'Best Model' from this homework does not need to be the one I have with highest score on leader board. It is the one I learned the most, which is the final model I ran before decided to stop and write this report. Even though it is not the best model statistically, I have made so many trials and errors, allow me to learn more about Keras and TensorFlow, specifically image classification. Therefore, I am happy with this assignment and all the models/knowledges I have gained and experimented throughout.