BBM 418 - Computer Vision Laboratory
Programming Assignment-3 Report

Spring 2020
Assoc. Prof. Dr. Nazlı İkizler Cinbiş

Arda Hüseyinoğlu
21627323

Department of Computer Engineering, Hacettepe University
May 22, 2020

1 Introduction

In this assignment, we will get familiar with image classification by training our own Convolutional Neural Network from scratch as well as using transfer learning methods. It is expected that we hold experiments to observe and analyze the effects of trying different parameters and different architectures to the prediction accuracy of the model on the given hand dataset. We also gain experience with PyTorch when implementing CNN.

In this report, there are two main sections. One of them is the implementation section which includes implementation and some coding details of the assignment. The other section includes all the experimental results and comments on them.

2 Implementation Details

Firstly, we check if the GPU is available to use. If so, we set device to GPU. In the data preprocessing stage, we need to read `'HandInfo.csv'` file to get label information for each image. `'gender'` and `'aspectOfHand'` columns are concatenated to create 8 class. After that, label encoder is used to convert these string type labels to integer type labels. Then, `'Hand'` folder which includes hand images is read and all the image file names are stored with their file path. After image paths and their corresponding labels are converted to dataframe, they are concatenated into a single dataframe.

After getting useful data, the data is split into train, validation and test. We use `train-test-split` method of the sklearn library when splitting. 15% of the data is used during testing, 17% of the data is used during validation and 68% of data is used during training. (Hand dataset consists of 11076 images. 7531 images are used in training, 1662 images is used in testing and 1883 images is used in validation.)

As the final process of the data preprocessing, we split data frame of each set (train, test, validation), which includes image path and label columns, into the two separate Pandas Series which are data labels and image paths to make data loader process easier.

After data preprocessing stage, we define our own specific dataset class which extends to the `'Dataset'` to be able to use them in data loader. When data loader brings the image-label pairs in chunks, it knows how to get it from the getitem method that is defined in `'HandDataset'` class. For each image-label pair, the given image path is read to get image and then image is converted RGB (since images are read by using `imread` method of Opencv) to work on same ordered color space. Then, some image transformations are applied to the image. In image transformation, the image is firstly converted to PIL image since some transformation methods can work with PIL image. Then the image is resized $1600x1200$ to $170x128$ and then converted to tensors. Lastly, it is normalized to get data within a range and to reduce the skewness which also decrease the training time. After image transformation, the image and its corresponding label is returned.

After creating data loaders for all sets, we need to define our network class to create our CNN model. All layers and their parameters (in-channel, out-channel, stride etc.) are defined (You can see the detail architecture in Experimental Results section) in the `'init'` method of the network class. Then we need to construct our CNN model in the forward method of the network class. After constructing all the layers, then we return the output by applying it softmax function to get prediction probabilities for each class. After initializing the model, 'cross entropy loss' and 'stochastic gradient descent' objects are created to use their functionalities in the model as loss function and optimizer.

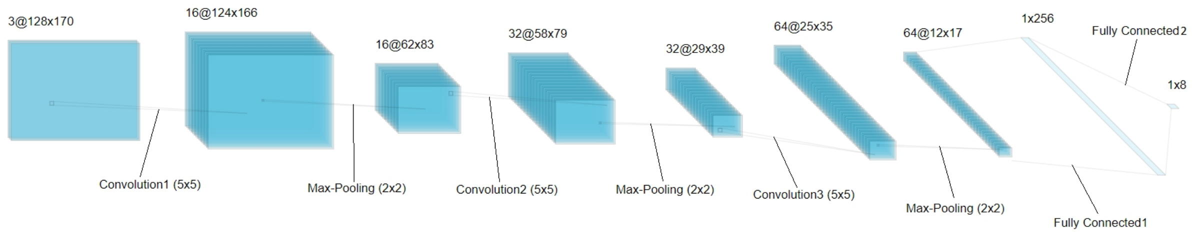
After defining the network, we can create the train and test methods. In train method, we do forward and backward propagation for 10 times (10 epoch) by using all the data in the training set. So, we use nested loops for that. In inner loop, data (image-label) is obtained chunk by chunk via train data loader. After there is no image to be brought by train data loader, which means that all training images did forward and backward propagation, we can proceed with next epoch and train the model again with all the training data, we can update our weights. Each loss that is calculated by cross entropy loss function is summed to get total loss after the final epoch. Validation loss is also calculated within the train method to get loss and accuracy change during training. When validation loss is calculated, there is no any backward propagation. Only thing to be done is that output (predictions) is calculated using trained network and validation image set. Then we get the loss value by applying cross entropy loss function. After getting loss value for validation and training, we can invoke the test method to get accuracy for each of them. Finally, we store loss values and accuracy percentage of each epoch to use them later when we plot loss and accuracy graph of the model (train method returns the list that includes those values).

In testing method, output (predictions) is calculated using trained network and testing image set. Predicted label is compared to the true label of the corresponding image. If it is predicted as true, number of 'correct predictions' variable increases by 1. Then, we can calculate the accuracy of the given dataset.

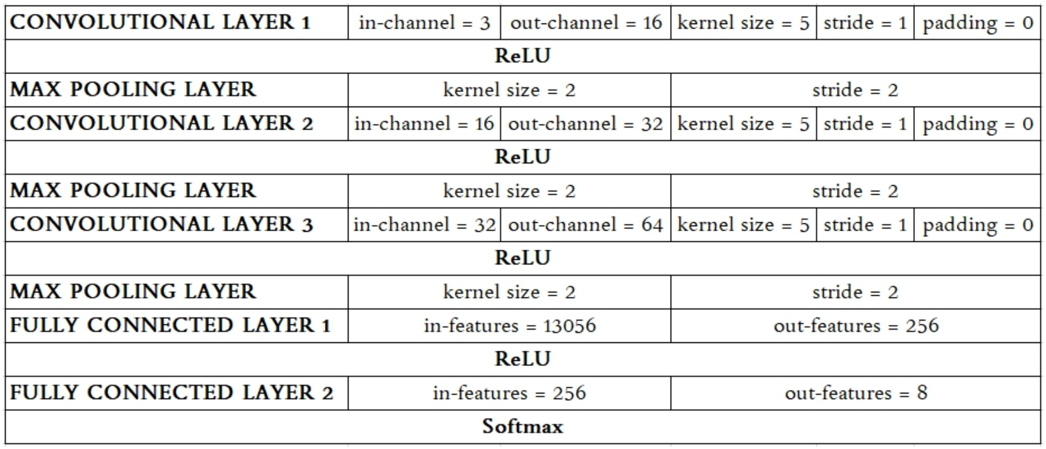
In part 2 of the assignment, which we use transfer learning when we train our model, only difference of the implementation is that we do not define and construct a network and just use pre-trained ResNet-18 model. Firstly, the pretrained model is obtained. After that, some layers of the network are frozen by changing their `'requires-grad'` attribute as 'False' in according to the experiment. We must change the last fully connected layer's out features as the number of class in our task which is 8 for every experiment arranged.

3 Experimental Results

3.1 Part 1



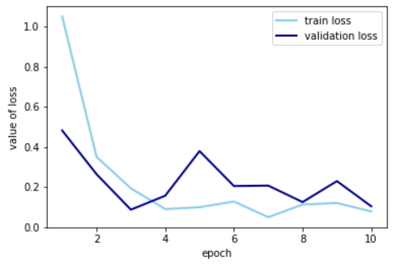
Initial architecture of my CNN model as follows: There are 3 convolutional layers, 2 fully connected layers and 3 max-pooling layers. Max-Pooling layers are applied after every convolutional layer. 'ReLU' is used as activation function. It is applied after every convolutional and fully connected layer - except the last fully connected layer which is also called classification layer. 'Cross Entropy Loss' is used as loss function and 'Stochastic Gradient Descent - SGD' algorithm is used as optimization algorithm in my CNN model. At the end of the network, 'softmax' function is applied to get probabilities belongs to each class.



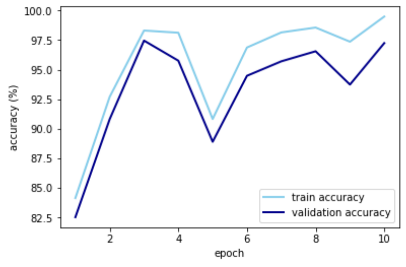
CONVOLUTIONAL LAYER 1	in-channel = 3	out-channel = 16	kernel size = 5	stride = 1	padding = 0
ReLU					
MAX POOLING LAYER	kernel size = 2		stride = 2		
CONVOLUTIONAL LAYER 2	in-channel = 16	out-channel = 32	kernel size = 5	stride = 1	padding = 0
ReLU					
MAX POOLING LAYER	kernel size = 2		stride = 2		
CONVOLUTIONAL LAYER 3	in-channel = 32	out-channel = 64	kernel size = 5	stride = 1	padding = 0
ReLU					
MAX POOLING LAYER	kernel size = 2		stride = 2		
FULLY CONNECTED LAYER 1	in-features = 13056		out-features = 256		
ReLU					
FULLY CONNECTED LAYER 2	in-features = 256		out-features = 8		
Softmax					

The following experiments are implemented when following hyperparameters are fixed:

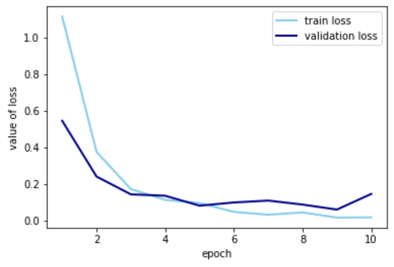
- Batch-size = 16
- Momentum = 0.9
- Number of Epochs = 10
- Input Image Size = Height: 128, Width: 170



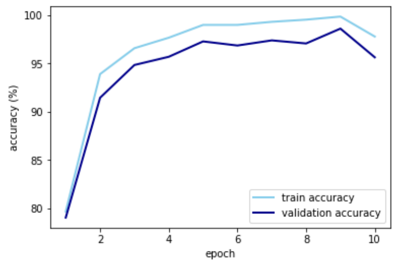
(a) Loss Change - Learning Rate = 0.01



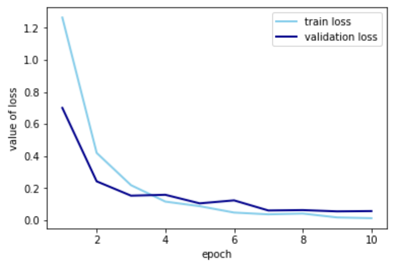
(b) Accuracy Change - Learning Rate = 0.01



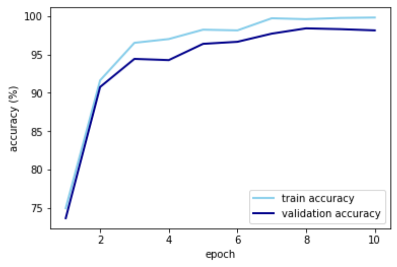
(c) Loss Change - Learning Rate = 0.005



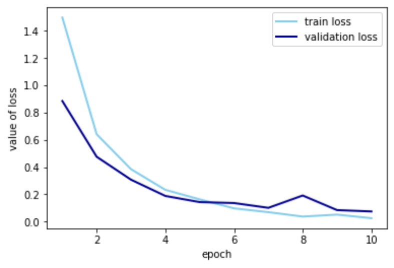
(d) Accuracy Change - Learning Rate = 0.005



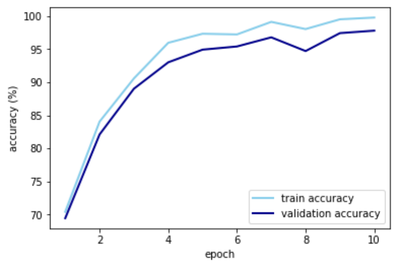
(e) Loss Change - Learning Rate = 0.0025



(f) Accuracy Change - Learning Rate = 0.0025



(g) Loss Change - Learning Rate = 0.001



(h) Accuracy Change - Learning Rate = 0.001

Accuracies on train and validation set after 10 epoch for 4 different learning rates:

Experiment 1: Learning Rate = 0.01

- Accuracy of the model on training set: 99.48214 %
- Accuracy of the model on validation set: 97.23845 %

Experiment 2: Learning Rate = 0.005

- Accuracy of the model on training set: 97.79578 %
- Accuracy of the model on validation set: 95.64525 %

Experiment 3: Learning Rate = 0.0025

- Accuracy of the model on training set: 99.81410 %
- Accuracy of the model on validation set: 98.14126 %

Experiment 4: Learning Rate = 0.001

- Accuracy of the model on training set: 99.74771 %
- Accuracy of the model on validation set: 97.76952 %

In according to the experiment results, I chose the model in experiment 3 whose learning rate is 0.0025 as the best hyperparameter due to best accuracy rate on validation dataset. Current best model's loss plot and accuracy plot both for training and validation set during training is given on figure(e) and figure(f).

- **Accuracy of the model on test set: 98.37545 %**
- Accuracy of the model on training set: 99.81410%
- Accuracy of the model on validation set: 98.14126%

Integration dropout to the network

Experiment 1: Dropout is only applied after first fully connected layer with $p = 0.4$

($\ldots$ →FC1 →RELU →**DROPOUT** ($p = 0.4$) →FC2 →$\ldots$)

- Accuracy of the model on training set: 98.45970 %
- Accuracy of the model on validation set: 95.80457 %

Experiment 2: Dropout is applied after the last max-pooling layer where we make the feature map flatten and after the first fully connected layer with $p = 0.4$ for both.

($\ldots$ →MAX-POOLING →**DROPOUT** ($p = 0.4$) →FC1 →RELU →**DROPOUT (**$p = 0.4$) →FC2 →$\ldots$)

- Accuracy of the model on training set: 97.63644 %
- Accuracy of the model on validation set: 96.49495 %

Experiment 3: Dropout is only applied after first fully connected layer with $p = 0.5$

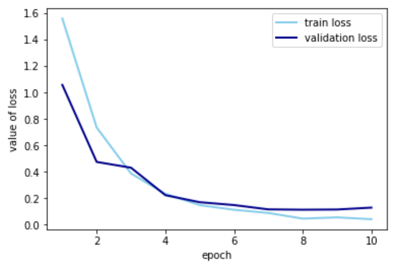
($\ldots$ →FC1 →RELU →**DROPOUT** ($p = 0.5$) →FC2 →$\ldots$)

- Accuracy of the model on training set: 98.24724 %
- Accuracy of the model on validation set: 96.22942 %

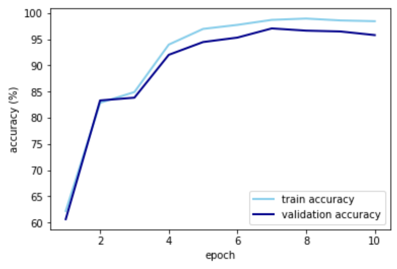
Experiment 4: Dropout is applied after first and second convolutional layer with $p = 0.2$ for both, and after first fully connected layer with $p = 0.5$

(CONV1 →RELU →**DROPOUT (p = 0.2)** →MAX-POOLING →CONV2 →RELU →**DROPOUT (p = 0.2)** →MAX-POOLING →CONV3 →RELU →MAX-POOLING →FC1 →RELU →**DROPOUT (p = 0.5**) →FC2 →$\ldots$)

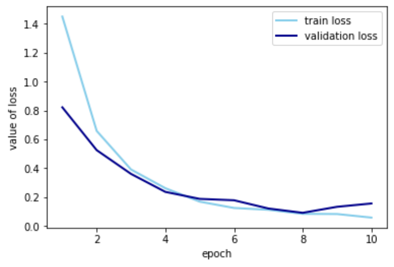
- Accuracy of the model on training set: 95.45877%
- Accuracy of the model on validation set: 93.57408 %



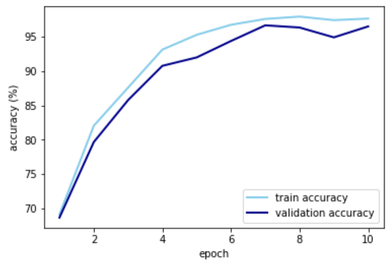
(a) Loss Change - Experiment 1



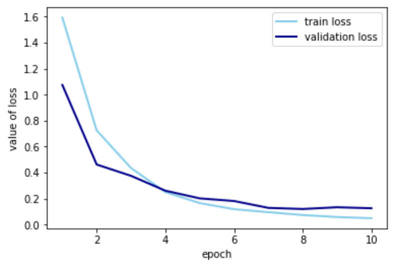
(b) Accuracy Change - Experiment 1



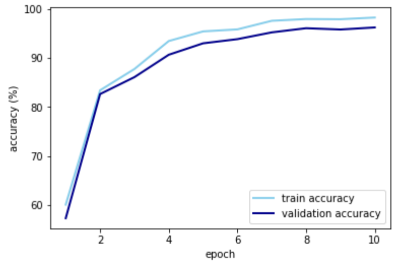
(c) Loss Change - Experiment 2



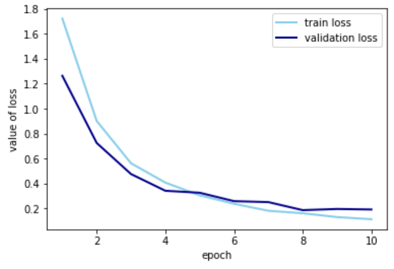
(d) Accuracy Change - Experiment 2



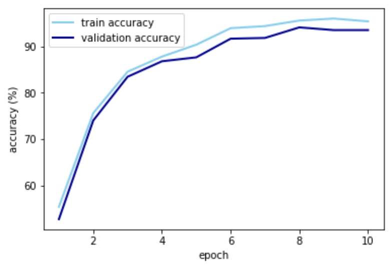
(e) Loss Change - Experiment 3



(f) Accuracy Change - Experiment 3



(g) Loss Change - Experiment 4



(h) Accuracy Change - Experiment 4

The above experiments are implemented when following hyperparameters are fixed:

- **Learning Rate = 0.0025**
- Batch-size = 16
- Momentum = 0.9
- Number of Epochs = 10
- Input Image Size = Height: 128, Width: 170

We can mention that the reason why the model in experiment 4 accuracy is less than the other models may be due to dropout layer after a convolutional layer cause loosing image features in an input image during training. Since we lose some information, our accuracy can decrease.

In according to the experiment results, I chose the model in experiment 2 as the best architecture due to best accuracy rate on validation dataset.

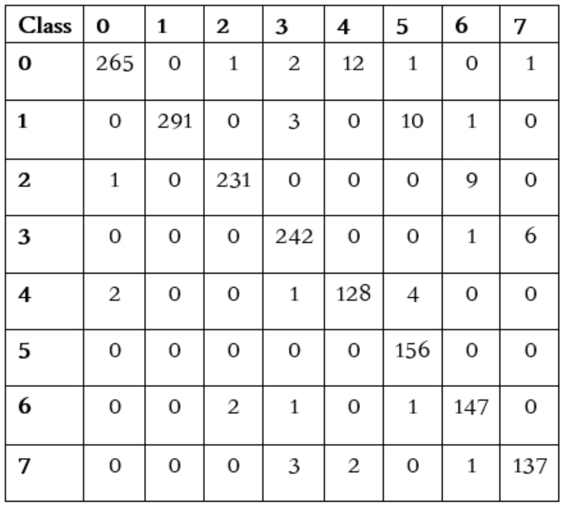
Finally, we get optimized architecture and best parameters in according to all the experiments:

- Learning Rate = 0.0025
- Batch-size = 16
- Momentum = 0.9
- Number of Epochs = 10
- Input Image Size = Height: 128, Width: 170
- **Architecture: -Model in Dropout Experiment 2-** (CONV1 →RELU →MAX-POOLING →CONV2 →RELU →MAX-POOLING →CONV3 →RELU →MAX-POOLING →DROPOUT (p = 0.4) →FC1 →RELU →DROPOUT (p = 0.4) →FC2 →SOFTMAX)

*Final model's loss plot and accuracy plot both for training and validation set during training is given on figure(c) and figure(d) where dropout experiment graphs are showed.

- **Accuracy of the final model on test images: 96.08905%**
- Accuracy of the final model on training set: 97.63644%
- Accuracy of the final model on validation set: 96.49495%

Confusion Matrix



Class	0	1	2	3	4	5	6	7
0	265	0	1	2	12	1	0	1
1	0	291	0	3	0	10	1	0
2	1	0	231	0	0	0	9	0
3	0	0	0	242	0	0	1	6
4	2	0	0	1	128	4	0	0
5	0	0	0	0	0	156	0	0
6	0	0	2	1	0	1	147	0
7	0	0	0	3	2	0	1	137

Classes and accuracy rates for each of them:

- 0: female dorsal left, accuracy : 93.97163%
- 1: female dorsal right, accuracy : 95.40983%
- 2: female palmar left, accuracy : 95.85062%
- 3: female palmar right, accuracy : 97.18875%
- 4: male dorsal left, accuracy : 94.81481%
- 5: male dorsal right, accuracy : 100%
- 6: male palmar left, accuracy : 97.35099%
- 7: male palmar right, accuracy : 95.80419%

The best classified class is ‘male dorsal right’ with 100% accuracy rate while the worst one is ‘female dorsal left’ with 93.97163% accuracy rate.

We can conclude that the model’s prediction on position of the hand (palmar-dorsal) and whether the picture contains right hand or left is generally correct except a few errors. However, the most mis-predictions is made on the ‘gender’ of the given hand image. (there are 65 images that is misclassified in total. 44 of them is misclassified due to ‘gender’)

We can also observe that the mentioned situation from the confusion matrix:

- 12 images belonging to '**female** dorsal left' class is predicted as '**male** dorsal left',
- 10 images belonging to '**female** dorsal right' class is predicted as '**male** dorsal right',
- 9 images belonging to '**female** palmar left' class is predicted as '**male** palmar left,
- 6 images belonging to '**female** palmar right' class is predicted as '**male** palmar right',
- 2 images belonging to '**male** dorsal left' class is predicted as '**female** dorsal left',
- 2 images belonging to '**male** palmar left' class is predicted as '**female** palmar left,
- 3 images belonging to '**male** palmar right' class is predicted as '**female** palmar right',

Finally, it can be said that the model is successful in general manner with its 96.08905% accuracy rate on the test dataset. But it can predict incorrectly the gender of the hand by the error rate of 0,02647%.

3.2 Part 2

What is fine-tuning?

Transfer learning basically enables us to use a neural network that is trained for one task and use it for another task. There are different types of scenarios of the 'fine-tuning' which can be used when transfer learning are applied.

In first case, we fine-tune / update all the weights in the layers of pre-trained model, which is ResNet-18 in our case, based on our new 'Hand' dataset by retraining the entire model. Since we do not freeze any layers in the model, which means that weights will be updated during backpropagation, we can optimize the entire features according to our 'Hand' dataset. That enables us to get better accuracy but increasing in training time is observed.

Another fine-tuning method is that the earlier layers are frozen and weight updating is only applied to some last layers of the network. Those last layers to be frozen may include convolutional layers besides fully connected layers in the model. The reason why that method works is that the earlier layers to be frozen in a network capture universal / low level features like curves and edges which are also relevant and useful to the most of the classification problems while later layers are more dataset specific. So we just update the weights of the our dataset-specific layers which is the last convolutional layers and fully connected layers.

Another method is that we can freeze all the layers except the last fully connected layer of the network which is the classification layer. This method is also called as feature extraction since we use pretrained ResNet-18 as a feature extractor.

Why should we do this?

One of the main reasons of using transfer learning is to make the training time shorter and to get better prediction results. It accelerates the training and improve the performance of a model for the specific task. For the performance perspective, it provides a higher starting point for the accuracy rate. The initial performance of the model is higher than the model that we do not use transfer learning (e.g. validation accuracies after first epoch). It also provides a faster improvement for the model during training. We can get more accurate results after even a few epochs in comparison with the model that we trained from scratch.

It is also good to use if our dataset size is small and we cannot find sufficient images to train the model from scratch– e.g. less than thousands. In deep learning, we need thousands or even millions of images (which depends on the task given) for training to get a good accuracy rate. If we have insufficient data to train our model from scratch, we can use these complex pre-trained models like ResNet-18 to get better results. In this way, we can gain the low and mid-level features from that pre-trained model.

Why do we freeze the rest and train only FC layers?

As the mentioned above, first layers capture low and mid-level features that are shared between images. So, we can freeze all the layers except FC layers, and use all those frozen layers as feature extractor and we can adapt the pretrained model to our new task by only retraining the weights of the fully connected layer (classification layer).

In other words, we do not need to train the weights of the convolutional layers and do not need to learn again how to extract features from scratch for our 'Hand' dataset since pre-trained model (ResNet-18) is used as feature extractor and it is complex enough. The reason why we need to train FC layer is that pre-trained model knows how to extract features but does not know how to classify them. So we need to retrain the weight of fully connected layer with our new 'Hand' dataset. Another reason why we only unfreeze the FC layer is that training time can also be shortened by this way and the method of fine-tuning all the layers or last of them can be time consuming during training.

Experiments

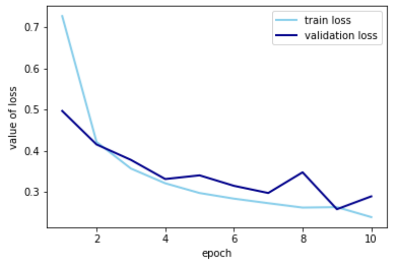
- Experiment 1 : Freezing all the layers except the last fully connected layer. (Validation accuracy -at the end of the 10th epoch- : 88.63516%)

- Experiment 2 : Unfreezing the last sequential layer (child 8) which includes two blocks that each of them have 2 convolutional layers, avgpool layer (child 9) and fc layer (child 10); and freezing the rest. (Validation accuracy -at the end of the 10th epoch- : 99.46893%

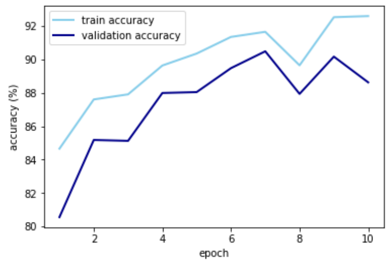
- Experiment 3 : Unfreezing all the layers, fine-tuning all the weights. (Validation accuracy -at the end of the 10th epoch-: 99.87966%)

The above experiments are implemented when following hyperparameters are fixed:

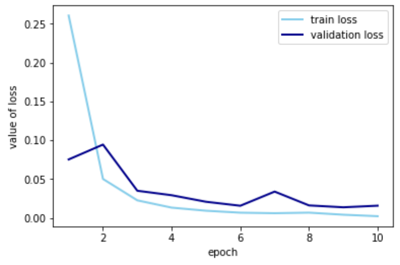
- Learning Rate = 0.0025
- Batch-size = 16
- Momentum = 0.9
- Number of Epochs = 10
- Input Image Size = Height: 128, Width: 170



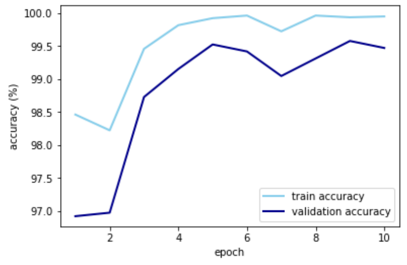
(a) Loss Change - Experiment 1



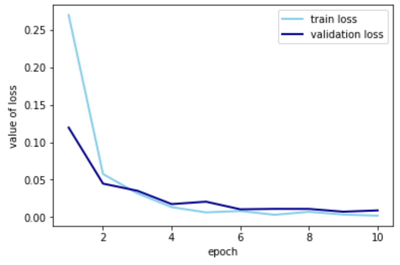
(b) Accuracy Change - Experiment 1



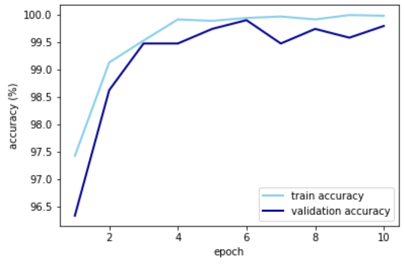
(c) Loss Change - Experiment 2



(d) Accuracy Change - Experiment 2



(e) Loss Change - Experiment 3



(f) Accuracy Change - Experiment 3

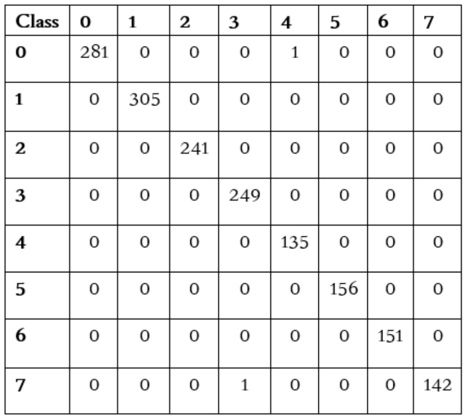
In according to the results of the experiment, unfreezing all the layers by fine-tuning all the weights in the model gave the best accuracy rate in comparison with the other models. There is too little accuracy difference between model 2 and model 3. The reasons of that small difference (0.41073) may be the differences of the frozen layers. In other words, earlier convolutional layers can capture low and mid-level features which is also relevant to our 'Hand' dataset. So, to update these layers will not cause a big difference when we use these layers as feature extractor (Model 3 updates these layers while model 2 do not update in experiment.) It is also observed that the training time difference between model 2 and model 3 is not so significant. That case may be due to the gradient of the layers to be updated is already close to minimum value thanks to pre-trained complex ResNet-18 architecture. So, it will be enough to do a couple of backpropagations with small learning rate when adapting our dataset to model. That can be the reason why is that there is no such big difference in the training time between model 2 and model 3.

The validation accuracy of the model 1 is less than the other two models. The reason of that case may be due to the last convolutional layers capture more dataset-specific features and we freeze all the layers except the fc layer. So, the layers except fc layer to be used as feature extractor will not be so specific to our task. In model2 and model3, we optimize the features according to our 'Hand' dataset.

Another observation is that higher starting point for the accuracy rate for all the models after first epoch. (Validation accuracies after 1st epoch $\rightarrow$Model 1: 80.56293%, Model 2: 96.91981%, Model 3: 96.33563%). We can easily observe that using transfer learning with fine-tuning method (unfreeze all the layers) improves the performance significantly. Computation time when training can be counted as the only disadvantage of that method in according to the experiments.

Finally, best trained model is the model 3 whose accuracy in test set is 99.87966

Confusion Matrix



Class	0	1	2	3	4	5	6	7
0	281	0	0	0	1	0	0	0
1	0	305	0	0	0	0	0	0
2	0	0	241	0	0	0	0	0
3	0	0	0	249	0	0	0	0
4	0	0	0	0	135	0	0	0
5	0	0	0	0	0	156	0	0
6	0	0	0	0	0	0	151	0
7	0	0	0	1	0	0	0	142

There are only 2 images that is misclassified. One is that the image belongs to 'female dorsal left' class but it is predicted as 'male dorsal left'. The other one is that the image belongs to 'male palmar right' class but it is predicted as 'female palmar right'. We can draw a conclusion that the model predict incorrectly the gender of the hand by the error rate of 0,0012 while it predicts the position of the hand (palmar-dorsal) and whether the picture contains right hand or left hand 100% correctly.

Model Improvement Ideas

Firstly, we can improve the performance by adding more data using image augmentation strategy. Thanks to image augmentation, we can increase the number of training images. We take the images in our current training dataset and apply some image transformation operations to them - e.g. rotation, shearing, translation. By this way, we can get more different training data.

Secondly, we can decrease the training time by applying batch normalization like we do when transforming images as input. In this way, the activations of hidden layers are normalized we make in the same distribution.

The other method that helps to improve the model's performance can be decreasing the learning rate. Since we use the pre-trained model in transfer learning, weights are already trained and there is no need to do so much backpropagation to minimize the gradients – this case is especially acceptable for the first convolutional layers which includes universal features. If we used bigger learning rates, then we would not reach the minimum and our training time would increase. So, as its name signifies, we should use smaller learning rates to 'fine-tune' the model. In this way, we can get more accurate predictions despite the computation time increases. Plus, in according to the experiment that I have done and the articles that I have read, to use transfer learning with fine-tuning all the layers gives the best performance.

Except the other improvement strategies mentioned above, we can reconsider the followings by doing an experiment again: tuning the dropout rate – applying dropout to different places to avoid overfitting, adding or reducing convolutional and fully connected layers with different parameter (e.g. stride, padding, number of neurons etc.), changing input image size (make it bigger with high resolution), increasing the number of epochs or tuning the batch size.

Comparison of the results in Part 1 and Part 2

Firstly, accuracy considerably increases when using transfer learning. After 10th epoch, validation accuracy of the model in part 1 (training the our own model from scratch) is 96.49495% and test set accuracy is 96.08905%; while validation accuracy of the model in part 2 (with transfer learning using pre-trained model) is 99.78757% and test set accuracy is 99.87966% in according to the experiments. However, I could not observe that the training time decreases when using transfer learning. Although training time in part 2 is longer than the part 1, there is no significant time difference between them. So, we can conclude that the using transfer learning with fine-tuning help us to get better prediction results.

We can also mention that, in part 2 where transfer learning method used has a higher starting point for the accuracy rate. After the first epoch, validation accuracy of the model in part 1 is 68.66702% while validation accuracy of the model in part 2 is 96.33563%. The initial performance of the model is higher when we use transfer learning. We can observe that transfer learning also provides a faster improvement for the model during training. We get more accurate predictions after a few epochs in comparison with the model in part 1. For example, after 3rd epoch, we already get 99.46893% accuracy on validation set for model in part 2, while we get 85.76739% accuracy for model in part 1.

Lastly, confusion matrices of both models show that: the most mis-predicted attribute of the hand image is being 'gender'. While the only error originates from the gender prediction with 0,00120% error rate for model in part2, most of the errors originate from the gender prediction with 0,02647% error rate.

4 References

- https://www.youtube.com/playlist?list=PLQVvvaa0QuDdeMyHEYc0gxFpYwHY2Qfdh
- https://debuggercafe.com/custom-dataset-and-dataloader-in-pytorch/
- https://medium.com/@thevatsalsaglani/multi-class-image-classification-using-cnn-over-pytorch-and-the-basics-of-cnn-fdf425a11dc0
- https://www.kaggle.com/danielhavir/pytorch-dataloader
- https://towardsdatascience.com/a-beginners-tutorial-on-building-an-ai-image-classifier-using-pytorch-6f85cb69cba7
- https://stackoverflow.com/questions/53290306/confusion-matrix-and-test-accuracy-for-pytorch-transfer-learning-tutorial
- http://alexlenail.me/NN-SVG/LeNet.html
- https://www.youtube.com/watch?v=K0lWSB2QoIQ
- https://discuss.pytorch.org/t/how-the-pytorch-freeze-network-in-some-layers-only-the-rest-of-the-training/7088/2
- https://stats.stackexchange.com/questions/240305/where-should-i-place-dropout-layers-in-a-neural-network