Classification Using Convolutional Neural Networks

AER1515 - Perception for Robotics

Assignment 1

October 14, 2022

Hsuan-Ling (Celene) Chen

0 Introduction

The convolutional neural network (CNN) represents a huge breakthrough in image recognition and is frequently used in image classification tasks. In this assignment, a CNN is used for an animal classification task. The images of animal faces are from the LHI-Animal-Faces dataset. It consists of 20 classes, 19 classes of different animals, and one human face class. There are a total of 2313 training images and 100 test images.

1.0 Run and Extend to Multi-Class Classification

To start, the start code is used to train a binary classifier on "Dog" and "Cat" using the CNN model and an evaluation code for testing the trained model. The training model is conducted on the training set of 2313 images. When training is done, it will generate a file called "model.pt" and a plot for the training loss . The trained model is then tested with the evaluation code that will report the overall accuracy result (%) on the test set of 100 images.

1.1 Plot of Training Loss

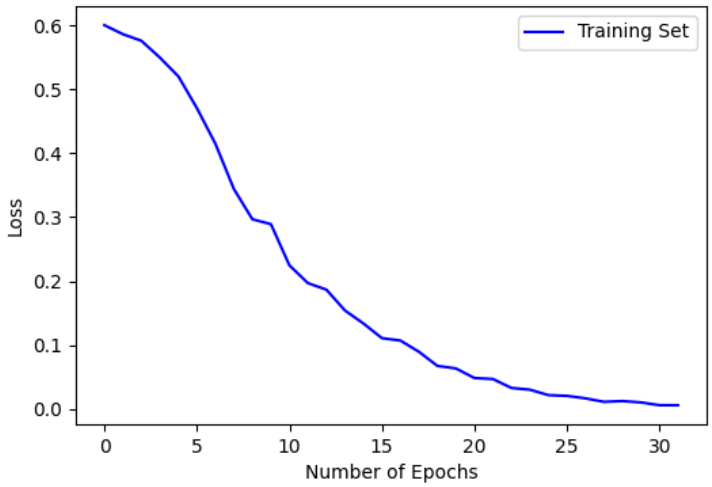

Figure 1: Training Loss Plot of binary-class classification task after 32 epochs

Figure 1 displays the loss plot of the CNN model, which was run with 32 epochs on binary classification on two classes. As the number of epochs iterations increases, the loss decreases over time, this is an indication that the model is learning.

1.2 Overall Classification Accuracy

Executing the evaluation code on the testing set, the overall classification accuracy is 80.0% on 10 test samples.

1.3 Extend Binary classification to a Multi-class classification

The binary classification is extended to a multi-class classification task in order to classify more classes of animals. Binary classification is the task where examples are assigned exactly one of two classes, which are “Dogs” and “Cats” in this case. Multi-class classification is the task where examples are assigned for more than two classes. The number of classes for the model to train has changed to 20. Without other modification to the

code, the resulting classification accuracy after 32 epochs is 63.0% on 100 test samples and its training loss plot is displayed in Figure 2.

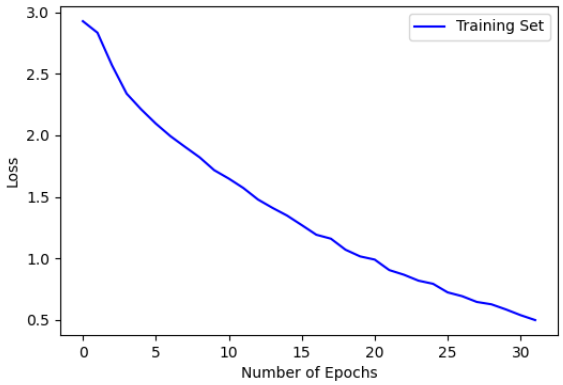

Figure 2: Training Loss Plot of 20 class multi-class classification task

2.0 Change CNN Architecture

The current CNN architecture consists of 4 convolutional layers, 3 max-pooling layers, and 2 fully connected layers with Rectified Linear Unit (ReLU) as the activation function. Additional layers need to be added to the current CNN architecture in order to improve the performance.

2.1 Batch Normalization Layer

Batch normalization is a layer that allows network layers to do learning more independently by normalizing the output of the previous layers. This helps the learning to be more efficient and offers regularization to avoid overfitting of the model. The goal of this layer is to achieve a stable distribution of activation values throughout training, hence, it is often placed just after the convolution and pooling layers but before ReLU as mentioned in the Batch Normalization paper. The parameter required by batch normalization is the number of out-channels features of the layer above it. [1] Figure 3 below shows the training loss plot of the modified CNN architecture after adding Batch Normalization to the CNN architecture with the overall classification accuracy of 77% with 100 test samples.

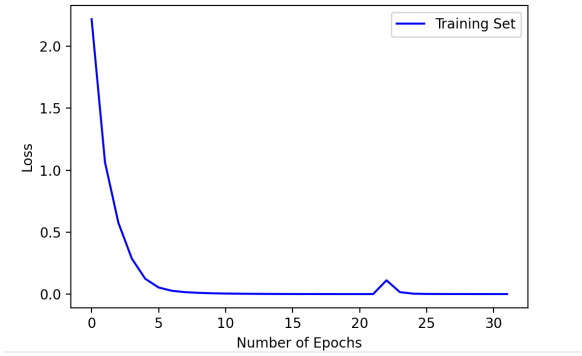

Figure 3: Training loss plot after adding Batch Normalization to the CNN architecture

2.2 Dropout Layer

Dropouts are another regularization technique that is used to prevent overfitting in the model. These layers are added to switch some percentage of neurons in the network off the incoming and outgoing connection. They

are usually avoided after the convolution layers, and are mostly used after the dense layers of the network. In this modified CNN architecture, 50% of the neurons are switched off between the convolution layers which have a high number of parameters so that the model will have less possibility of overfitting and memorizing the training data. Figure 4 shows the resulting training loss plot and the test accuracy of 78.0% is achieved on 100 test samples after adding the dropout layers to the CNN architecture.

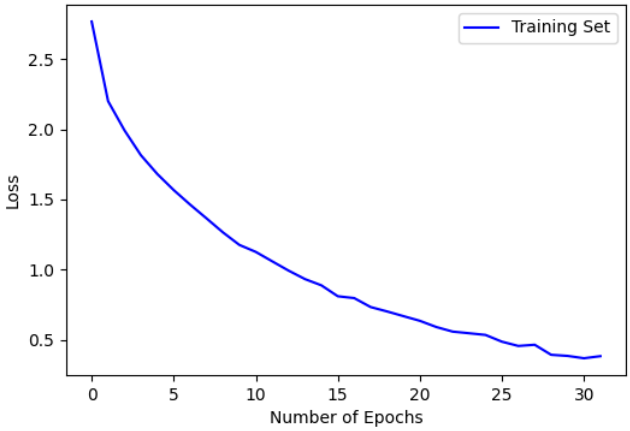

Figure 4: Training loss plot after adding Dropout to the CNN architecture

3.0 Training Neural Network with Validation

In the start code, the full training data is split into the training set (80%) and validation set (20%). So far, only the training set is used and the validation set has not been leveraged yet. The performance of the model can be further optimized with the use of the validation set. Moreover, the epoch which has the least validation loss can be kept track of and saved as the best model at the end of the training.

3.1 Validation Loop

To train the neural network with validation, the loop is the same as the training loop but with forward pass and loss calculation only. The training set learning curve is calculated from the training dataset which shows how well the learning, the validation on the other hand, shows how well the model generalizes.

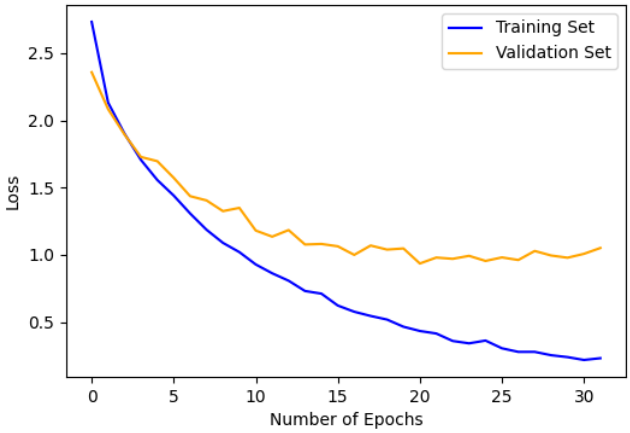

Figure 5: Training and Validation Loss plot

Figure 5 shows the plot that includes both training loss and validation loss with 32 epochs of the modified CNN architecture.

3.2 Evaluation

The CNN model that corresponds to the least validation loss is saved to run the evaluation on. The new classification accuracy with the validation set is 76.0% on 100 test samplers as compared to the model, which had 78%, that does not use the validation set. The validation set is used to provide unbiased evaluation of the model fit on samples that were not used to build the model. It is useful for hyper-parameter tuning and model selection. For example, for every epoch, as the model is training with the training set, the model evaluation is performed on the validation set. This helped to keep the model in check to not overfit as it can become very good at classifying samples in the training but cannot generalize and make accurate classifications on the data it has not seen before. Hence, the classification accuracy with the validation set is slightly lower because the evaluation results of the previous model without the validation set is biased and misleading with respect to how well the model is able to generalize to new data.

4.0 Hyperparameter Tuning

Up to this section, the trained CNN model has been using the RMSProp as the optimizer for the network training. It uses a batch size of 32, and a learning rate of $1\times e{-}4$. Hence, in this section, the tuning and discovery of new combinations of hyperparameters are explored to improve the classification result on the test set.

4.1 Different Optimizers

Optimizers are used to tune the parameters of a neural network in order to minimize the cost function. The torch.optim used in this assignment is a package that uses various optimization algorithms. Two of the more commonly used optimizers Adam and SDG were being tested on the modified CNN architecture, below are their classification results.

Table 1: Experimental results by various Optimizers

Optimizer	**Classification Accuracy**	**Number of test samples**
RMSProp	76.0%	100
SDG	6.0%	100
Adam	77.0%	100

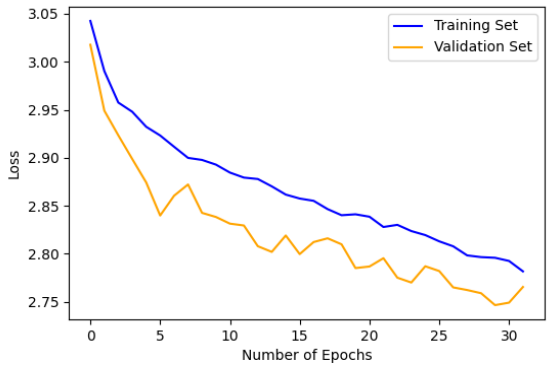

Figure 6: Training loss plot using SDG optimizer

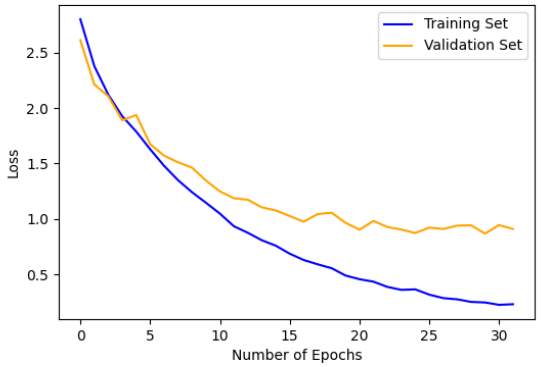

Figure 7: Training loss plot using Adam optimizer

It can be seen from the results that the Adam optimizer did only slightly a better job in accuracy than SDG and RMSProp. The training results for the SDG optimizer resulted in a very bad model of only 6% accuracy in this application.

4.2 Learning Rate

From the results of the above section, Adam optimizer is now selected to be the new optimizer for the CNN architecture. The learning rate is tuned for the training, the training and validation loss plots with the different learning rates and accuracy are shown below.

Table 2: Experimental results by various Learning rate

Learning Rate	**Test Accuracy**	**Number of test samples**
1e-2	61.0%	100
1e-3	76.0%	100
1e-4	77.0%	100
1e-5	41.0%	100

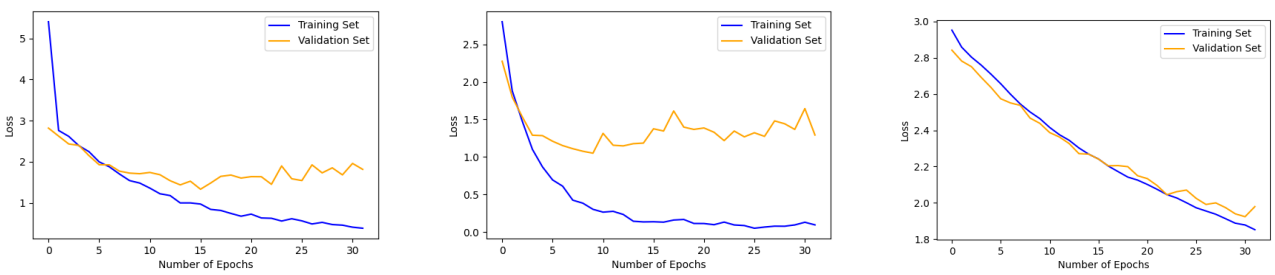

Figure 8: Training and Validation Loss plots with learning rates 1e-2, 1e-3, 1e-5 respectively

The learning rate of the model regulates the amount of allocated error with which the model's weights are updated each time they are updated. A desirable learning rate should be low enough for the network to

converge on something useful while still high enough to train in a reasonable length of time. Smaller learning rates need more training epochs because of the fewer changes. On the other hand, larger learning rates result in faster changes but can result in a suboptimal final set of weights.
As seen from the plots in figure 8, the plots displayed an oscillation behavior when the learning rate is large at 1e-2 and 1e-3, at smaller learning rate of 1e-5, the model becomes slower at learning to converge. The model is set up with a learning rate to remain at 1e-4 as the final decision since it shows the best performance out of the other learning rates.

4.3 Batch Size

Continuing with the current modified CNN architecture, the batch size can be tuned in order to further optimize the mode. The batch size defines the number of samples that are passed to the network at once in a group. Therefore, larger batch size will yield quicker trained models but with a tradeoff of lower quality models since the model will be unable to generalize well on data it has not seen before. However, models trained with smaller batch size can take longer time to train and do not necessarily yield better training accuracy since they are exposed to more randomness in each batch of dataset. Hence the appropriate batch size needs to be tested and tuned based on the model and its resource utilization. In this project, there are 1850training samples, batch sizes of 8, 16, 20, 32 and 64 are selected to test with results shown in following Table 3 and Figure 8.

Table 3: Experimental results by various batch sizes

Batch Size	**Test Accuracy**	**Number of test samples**
8	76.0%	100
16	80.0%	100
20	79.0%	100
32	77.0%	100
64	74.0%	100

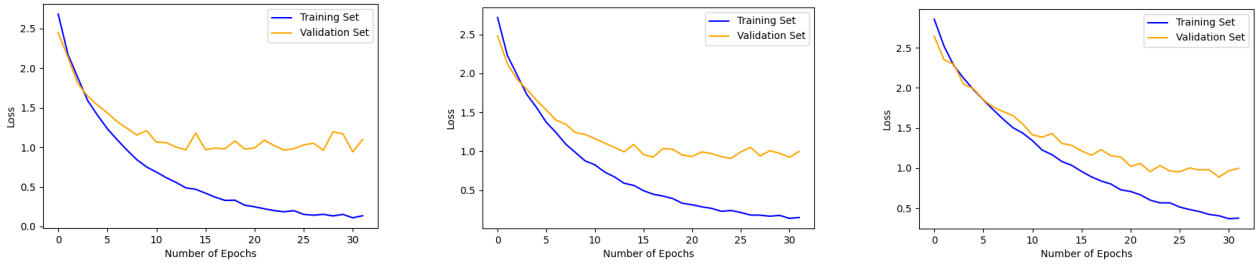

Figure 8: Training and Validation Loss plots with batch sizes 8, 16, 64 respectively

From the results and loss plots, it can be seen that the smaller batch sizes yields a higher test accuracy and the models trained with small batch sizes generalize better on the validation set.

4.4 Comparisons of Different Combinations of Hyperparameters

Using all the experimental results from the previous sections on optimizer type, learning rate, batch size and their effects, a combination of tuning and optimization can be achieved for a better trained model on the CNN architecture. After some experimental testings and tweaking of the hyperparameter values, the best model with this modified CNN architecture achieved is 81.0% with the following hyperparameters:

Optimizer: Adam
Learning Rate: 1e-4
Batch Size: 16
Number of Epochs: 50
Validation Percentage: 0.2

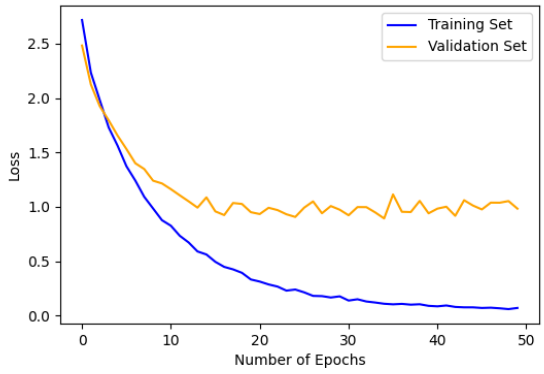

Figure 9: Training and Validation Loss plot of final optimized model

5.0 Conclusion

In conclusion, it can be seen that sufficient accuracy of 81.0% can be met on the trained CNN model for 20 class classification after a series of hyperparameters tuning. Models can be improved by further experimenting with the choice of optimizer, learning rate, batch size, number of epochs and validation percentage.

Reference

[1] Admin. (2021, August 29). *How to use the BATCHNORM layer in pytorch?* Knowledge Transfer. Retrieved October 14, 2022, from https://androidkt.com/use-the-batchnorm-layer-in-pytorch/