Project Image Recognition: Monkey Species Classifier

Introduction

Computer vision algorithms have seen great innovations over the past few years. These innovations in algorithms design have been applied to greatly improve perception ability in applications such as self-driving cars. Within computer vision, there are two general categories of classification tasks: general classification and fine-grain classification. A general classification task involves perceiving images and classifying them as one among a set of classes such as cats or dogs. A fine grain classification task involves classifying images as one amongst different species of sparrows. Theoretically, this classification task would require the neural network learning a more complex and detailed understanding of the image data. In a neural parallel, humans are able to perform this type of classification with ease after seeing a few examples.

A detailed fine-grain classification approach holds great promise for autonomous systems and image recognition. A self-driving car or delivery drone that can perceive the greater difference between types of objects could develop a learned association with that object type, enabling it to respond to a greater range of situations. In addition, efficient fine-grain classification algorithms would give any artificial agent a more detailed and accurate perception of its environment.

At a simpler and more applied level, this project solves a fine-grain classification task using a fully defined CNN. The task is to predict the species of a monkey in the image.

Data

The data for this project was obtained from Kaggle.com. The data includes a training and a validation dataset of which the training set was split further into a training and test dataset. Each dataset includes images from the ten different classes. The images are of varying dimensions and required standardizing through PyTorch transforms. More information on the dataset can be found at the Kaggle dataset page.

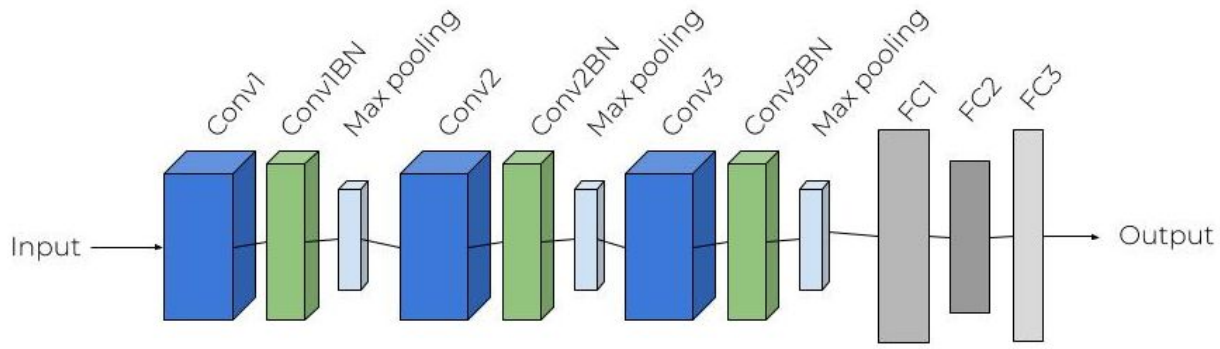
Kaggle Link: https://www.kaggle.com/slothkong/10-monkey-species

The total number of images in the training dataset was 1097. Interestingly, 1097 is a prime number, resulting in uneven batch sizes. Since the number of images per class was roughly the same, a random image was removed to create a total training set size of 1096. This allowed for a batch size of 137 per training dataset. For the test dataset, a random subset totalling 25% of the 1096 training images was isolated. In total, the training dataset included 822 images, the test data included 274 images, and the validation data included 274 images.

CNN Model

The CNN model used in this project consists of 3 convolutional layers, 3 batch normalization layers, and 3 fully connected layers with dropout. The architecture is visualized below.

Figure 1: CNN Architecture



The abbreviations in the figure are as follows Conv1 is Convolutional layer 1, Conv1BN is Batch Normalization layer of the outputs of Convolutional layer 1, and FC1 is fully connected layer 1. The specific convolutions are mentioned below.

Conv1: 2D convolution with input channels=3, output channels=16, kernel size=4, padding=1, stride=2
Conv2: 2D convolution with input channels=16, output channels=32, kernel size=3, padding=1
Conv3: 2D convolution with input channels=32, output channels=64, kernel size=4, padding=1
FC1: Fully-connected linear layer with an input of 4096 and an output of 100
FC2: Fully-connected linear layer with an input of 100 and an output of 1000
FC3: Fully-connected linear layer with an input of 1000 and an output of 10

The outputs of a convolutional layer are fed into a batch normalization layer. The outputs of the batch normalization layer are the input into the max pooling layer. The final pooling layer's output is the input for the first fully connected layer. Each output of a FC layer is filtered using a ReLU function and then into a dropout function. The output of FC3 is the input for a log softmax function, which returns the final output of the network.

Data Preprocessing

The training data was transformed using PyTorch's transforms. To augment the data, the training image data was randomly horizontally flipped and/or randomly vertically flipped with a probability of 0.3. The images were normalized to a range of [-2,2] instead of [-1,1]. The larger normalization increases the possible range of values

for image pixels. This particular monkey species dataset contains some species with very similar image pixel values and some species with drastically differing attributes. In theory, the similar features can be better distinguished and the differing attributes can be further separated with a wider range of values. A wider normalization was chosen so that the increased range of possible values would allow for greater differentiation by the model for different image pixel values.

Training the Model

The model is trained on the training data described earlier. The model architecture was manipulated and evolved through repeated experimentation. After arriving on a generally well-performing architecture, the learning rate was adjusted to obtain optimal accuracy.

The model weights were saved at epoch 45. The criterion used was Cross Entropy Loss. The optimizer used was the Adam optimizer. The network architecture was consistent between trials with different learning rates. The accuracy on a test dataset and the corresponding learning rate for four different trials is presented below.

Trial 1: Learning Rate = 0.01	Test Accuracy = 6%
Trial 2: Learning Rate = 0.001	Test Accuracy = 69%
Trial 3: Learning Rate = 0.0001	Test Accuracy = 66%
Trial 4: Learning Rate = 0.00001	Test Accuracy = 30%

Further analysis was required to find the optimal hyperparameter values for epochs and learning rate. From these four trials, an initial direction for tweaking the learning rate to obtain the highest performance is a learning rate between 0.0001-0.001. To further test this approach, three further rates were tested.

Trial 5: Learning Rate = 0.0003	Test Accuracy = 67%
Trial 6: Learning Rate = 0.0005	Test Accuracy = 66%
Trial 7: Learning Rate = 0.0008	Test Accuracy = 68%

The training and validation loss for Trials 1-4 and 5-7 are plotted in Figure 2 and 3, respectively.

Figure 2: Training and Validation Loss over 45 Epochs for Trials 1-4

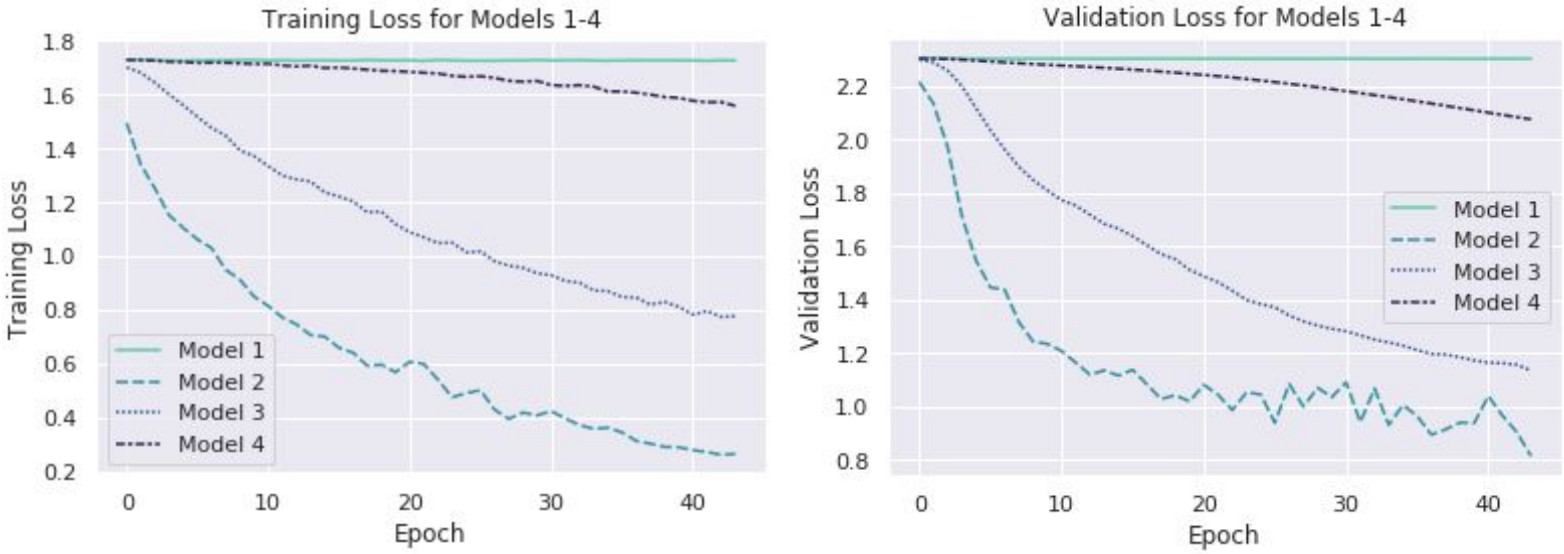
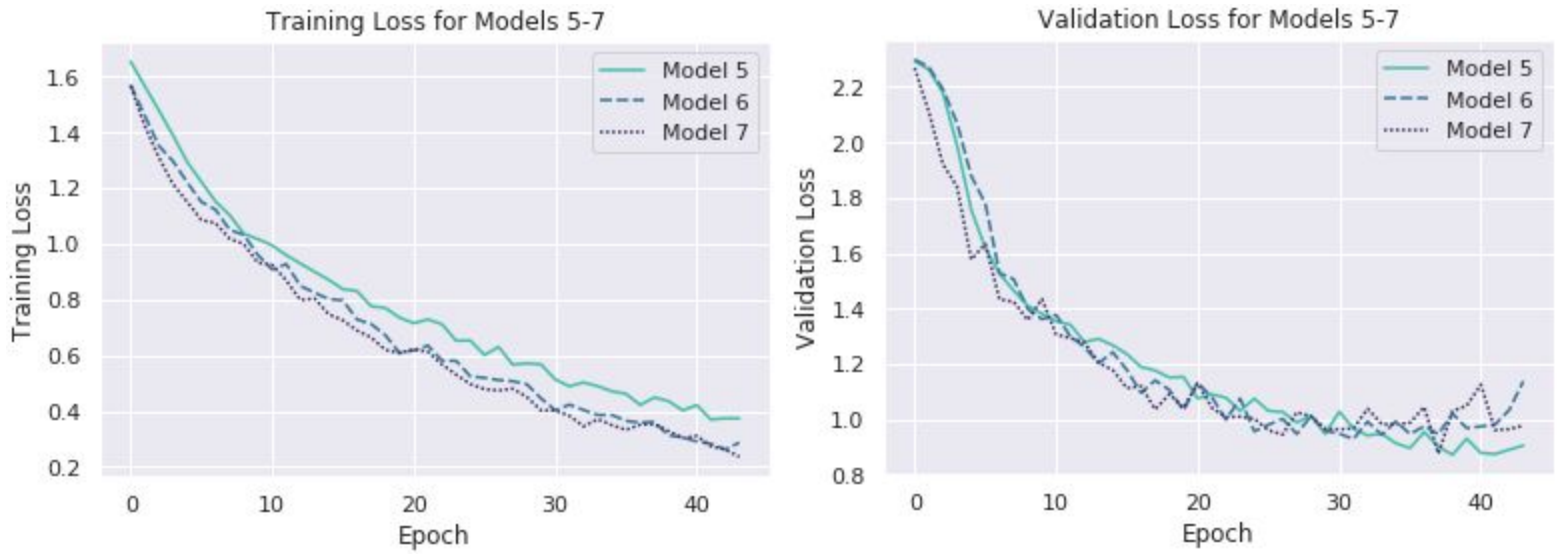


Figure 3: Training and Validation Loss over 45 Epochs for Trials 5-7



From the test accuracies and the training and validation losses, it appears the optimal value for the learning rate is 0.0008. The accuracy on this test set is highest for that learning rate. The validation loss beginning to increase around epoch 30 could indicate that the higher performance on this test set may be due to some degree of overfitting. Further reruns of the learning rate of 0.0008 indicate that stopping at an earlier epoch increases the test accuracy and reduces the trend of increasing validation loss.

By stopping at epoch 34, the model achieves an accuracy of 72% on test set. The model weights learned at this epoch are before the validation loss begins

increasing, suggesting that this model avoids the overfitting of the model trained until epoch 45. The per class accuracy is reported in Table 1 below

Table 1: Optimal Model Performance on Test Dataset

Class	#Correct/Total	Class Accuracy
Species 1	15/20	75%
Species 2	18/22	81%
Species 3	25/30	83%
Species 4	20/31	64%
Species 5	26/39	66%
Species 6	21/24	87%
Species 7	17/23	73%
Species 8	18/29	62%
Species 9	22/27	81%
Species 10	15/27	55%
Total	**197/272**	**72%**

Conclusions

The optimal model for this CNN architecture is one that learns at a learning rate of 0.0008 for 34 epochs. The model architecture is designed with repeated batch normalization after each layer and dropout to enable robust learning. The lower training loss and relatively low validation loss suggest that the model has learned an accurate and relatively well fitting representation of the image features.

For fine-grain classification problems, the increased detail of the features and the numerous overlap of features between classes created a unique problem for the model to analyze. For example, all images of the ten monkey species shared many key features such as similar facial structure or environment as many of these images include branches and leaves. In this instance, the model must specifically learn to distinguish the differentiating features from the similar features. This architecture specifically includes batch normalization and dropout to build this learning ability. The wider normalization range also further differentiates the pixel values for similar features.

The repeated trials with varied learning rates allowed for targeted optimization for the network architecture and this fine grain classification task. While this architecture's features produced a relatively good performance after training, further architectural improvements such as bilinear CNNs could be better adapted for fine-grain classification tasks.