EE 5450 Project 02: Convolution Neural Network Classification of the Animals Data Set

David R. Mohler

April 18, 2018

1 Introduction

The rebirth of Neural Networks (NNs) and Machine Learning (ML) has offered revolutionary approaches to intelligent tasks previously unattainable in the field of image processing and computer vision. A major area of research continues to be the automated classification of images. Image classification through classical means of image processing tend to suffer due to problems with the wide variance in images to be classified and how their subject matter may be transformed from image to image. The discovery of Convolution Neural Networks (CNNs) lead to a massive increase in the accuracy of which images could be classified in an automated fashion.

For this project we will demonstrate the development of several convolution neural networks that vary across a wide number of parameters for the purpose of classifying color images of three different types of animals. The CNN is tasked with classifying images of cats, dogs, and pandas, typical examples of images in the data set can be seen below. Through the variation of network parameters such as the number of layers and the number of neurons within them, activation functions, learning rates, optimization routines, data augmentation, and other hyper parameters, we display the ability to increase the classification performance of the network.

2 Methods and Results

After establishing an initial understanding of the basic `ShallowNet` animal classification network that was provided, we begin with a relatively simple and small expansion to include fundamental elements of CNNs, once a working network is establish we are able to tune the parameters and observe the effects of varying the optimization techniques. Once a candidate optimizer has been selected we proceed to modify and tune the associated hyperparameters such as the learning rate (when applicable), batch size, number of training epochs, etc. Beyond this we also test the effects of data augmentation.

2.1 ShallowNet

For baseline comparisons we will use the code provided by Dr. Rosebrock [1] which is a simplistic implementation of a Convolution Neural Network designed with the goal of classification of images in to any of the three given animal categories. The basic structure of Rosebrock's network consists of:

`INPUT=>CONV=>RELU=>FC=>SOFTMAX`

Even with the most basic of implementations, the network is able to provide a respectable first attempt at the classification of the images. As seen in Table 1, the network achieves an average accuracy 66%. Given that with three classes the network has a 33% probability of randomly selecting the correct class, this is a decent first attempt. Using this as a lower bound we proceed to implement a number of other techniques that are common in CNN architectures in order to improve the network performance overall.

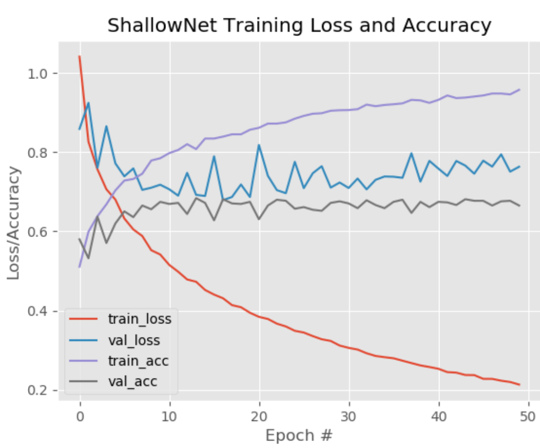

Figure 1: ShallowNet Accuracy

	Precision	Recall	F1-Score	Support
Cat	0.64	0.57	0.6	262
Dog	0.57	0.58	0.58	249
Panda	0.79	0.86	0.82	239
Avg/Total	0.66	0.67	0.66	750

Table 1: ShallowNet Results

2.2 MohlerNet2

The initial modification to the original network lies in the expansion of the architecture to incorporate key elements of CNNs, such as pooling layers and incorporation of neuron dropout prior to the fully connected layer. The general architecture of "`MohlerNet2`" is as follows:

```
INPUT=>CONV=>CONV=>MAXPOOL=>DROPOUT(0.5)=>FC=>SOFTMAX
```

Using the classical stochastic gradient descent (SGD) approach to the optimization of the network we were able to see an immediate and reproducible increase in the ability of the network to accurately classify the three categories of animal images. From Table 2, it can be seen that the inclusion of the additional convolutional layer, pooling, and dropout was able to provide a 6% increase in the accuracy of the network. For this experiment we used the keras recommended learning rate ($LR = 0.01$). As a more objective measure of the algorithm we can view the F-1 score for each tested network. The F-1 score captures information regarding the false positives and false negatives in the classification process, this gives a more objective view from algorithm to algorithm . From this we can also see a similar increase in F-1 score between the original ShallowNet implementation and the first implementation of MohlerNet (MohlerNet2).

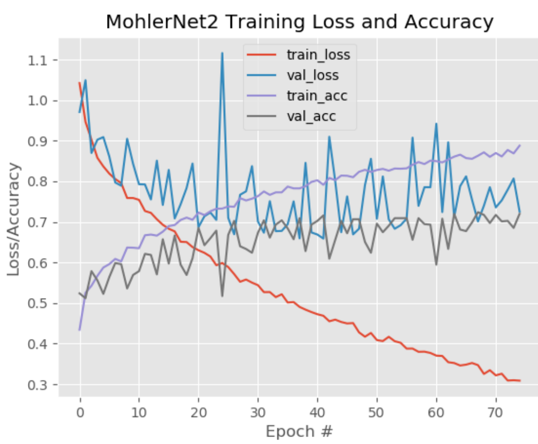

Figure 2: MohlerNet2 Accuracy, Learning Rate:0.01

	Precision	Recall	F1-Score	Support
Cat	0.67	0.75	0.71	262
Dog	0.64	0.60	0.62	249
Panda	0.88	0.82	0.85	239
Avg/Total	0.72	0.72	0.72	750

Table 2: MohlerNet2 Classification Results

Once the first results of the SGD optimized results were obtained we observed the effect of modifying the optimizer for the network. While using the AdaGrad and Adam optimzers, which apply adaptive learning rates, we saw no improvement in any of the scoring metrics. Actually, there was a reduction in accuracy from the original SGD approach, down to an average of approximately 70% classification accuracy.

TALK ABOUT OVERFITTING HERE

2.3 MohlerNet3

```
INPUT=>[CONV=>MAXPOOL]*2=>CONV*3=>MAXPOOL=>DROPOUT(0.5)=>FC=>SOFTMAX
```

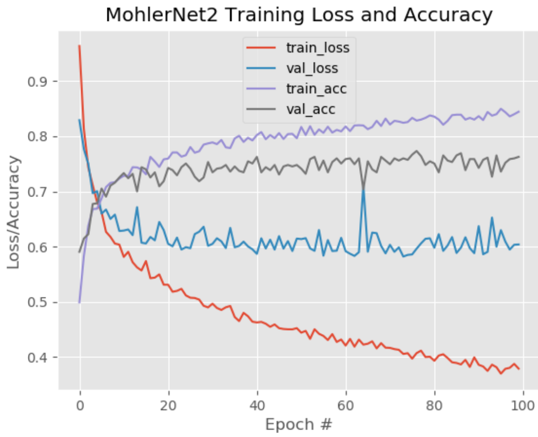

Figure 3: MohlerNet2 with Data Augmentation Accuracy, AdaGrad Optimized

	Precision	Recall	F1-Score	Support
Cat	0.75	0.66	0.71	262
Dog	0.69	0.7	0.69	249
Panda	0.84	0.93	0.89	239
Avg/Total	0.76	0.76	0.76	750

Table 3: MohlerNet2 with Data Augmentation Results

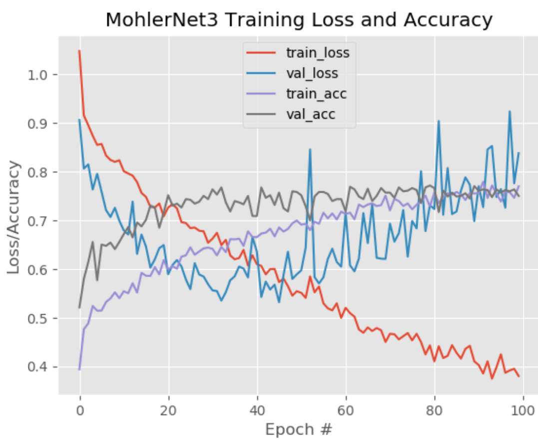

Figure 4: MohlerNet3 with Data Augmentation Accuracy, AdaMax Optimized

	Precision	Recall	F1-Score	Support
Cat	0.81	0.58	0.68	262
Dog	0.62	0.78	0.69	249
Panda	0.88	0.91	0.89	239
Avg/Total	0.77	0.75	0.75	750

Table 4: MohlerNet3 with Data Augmentation Results

2.4 MohlerNet4

```
INPUT=>[CONV=>MAXPOOL]*3=>FC=>DROPOUT=>FC=>SOFTMAX
```

3 Conclusions

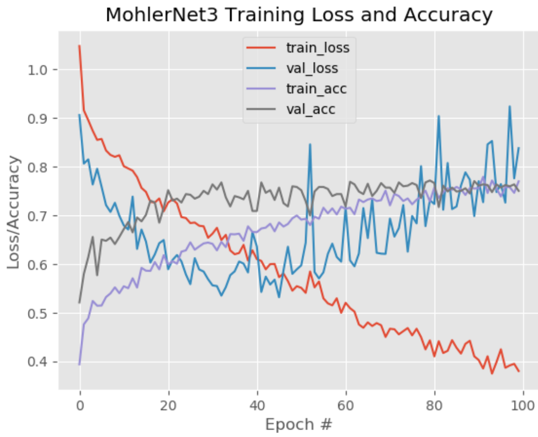

Figure 5: MohlerNet3 with Data Augmentation Accuracy, AdaMax Optimized

	Precision	Recall	F1-Score	Support
Cat	0.81	0.58	0.68	262
Dog	0.62	0.78	0.69	249
Panda	0.88	0.91	0.89	239
Avg/Total	0.77	0.75	0.75	750

Table 5: MohlerNet3 with Data Augmentation Results

A Code Listings

Listing 1: MohlerNet Implementations

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.convolutional import Conv2D,MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten,Dropout
from keras.layers.core import Dense
from keras import regularizers
from keras import backend as K

class MohlerNet1:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # ”channels last”
        model = Sequential()
        inputShape = (height, width, depth)

        # if we are using ”channels first”, update the input shape
        if K.image_data_format() == ”channels_first”:
            inputShape = (depth, height, width)

        # define the first (and only) CONV => RELU layer
        model.add(Conv2D(32, (3, 3), padding=”same”,
            input_shape=inputShape))
        model.add(Activation(”relu”))
```

```
        # softmax classifier
        model.add(Flatten())
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model

#ADD CONVOLUTION LAYER, POOLING LAYER, AND DROPOUT
class MohlerNet2:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last"
        model = Sequential()
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # define the first (and only) CONV => RELU layer
        model.add(Conv2D(32, (3, 3), padding="same", activation='
           [+]relu',
            input_shape=inputShape))
        #model.add(Activation("relu"))

        #ADD CONVOLUTION LAYER, POOLING LAYER, AND DROPOUT
        model.add(Conv2D(64,(3,3),padding="same",activation='relu'
           [+]))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.5))

        regularizers.l2
        # softmax classifier
        model.add(Flatten())
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model

class MohlerNet3:
    @staticmethod
```

```
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last"
        model = Sequential()
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # define the first (and only) CONV => RELU layer
        model.add(Conv2D(64, (3, 3), padding="same", activation='
           [+]relu',
            input_shape=inputShape))
        model.add(MaxPooling2D(pool_size=(2,2)))

        #ADD CONVOLUTION LAYER, POOLING LAYER, AND DROPOUT
        model.add(Conv2D(128,(3,3),padding="same",activation='relu
           [+]'))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Conv2D(64,(3,3),padding="same",activation='relu'
           [+]))
        model.add(Conv2D(32,(3,3),padding="same",activation='relu'
           [+]))
        model.add(Conv2D(32,(3,3),padding="same",activation='relu'
           [+]))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Flatten())
        model.add(Dense(classes))
        model.add(Dropout(0.5))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model


class MohlerNet4:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last"
        model = Sequential()
        inputShape = (height, width, depth)
```

```

        # if we are using "channels first", update the input shape
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # define the first (and only) CONV => RELU layer
        model.add(Conv2D(32, (3, 3), padding="same", activation='
           [+]relu',
            input_shape=inputShape))
        model.add(MaxPooling2D(pool_size=(2,2)))
        #model.add(Activation("relu"))

        #ADD CONVOLUTION LAYER, POOLING LAYER, AND DROPOUT
        model.add(Conv2D(32,(3,3),padding="same",activation='relu'
           [+]))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Conv2D(64,(3,3),padding="same",activation='relu'
           [+]))
        model.add(MaxPooling2D(pool_size=(2,2)))
        # softmax classifier
        model.add(Flatten())
        model.add(Dense(64))
        model.add(Activation("relu"))
        model.add(Dropout(0.5))
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

References

[1] A. Rosebrock, *Deep Learning For Computer Vision With Python*. PyImageSearch, 2017.