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

David R. Mohler

April 19, 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 tends 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 amount 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 established we are able to tune its 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.

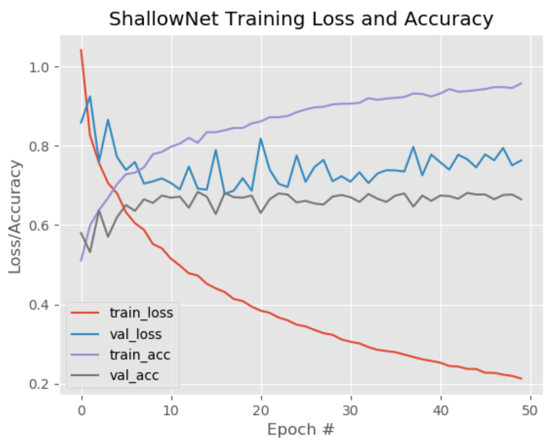

Figure 1: ShallowNet Accuracy

	Precision	Recall	F1	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/Tot	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 data regularization through 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
```

More specifically, the CNN employs the use of 3×3 kernels in the convolution layers, where the first layer generates 32 feature maps and the second expands to 64 feature maps prior to applying a 2×2 max-pooling to the output. From the pooled output a neuron dropout probability of 50% is applied in order to attempt to reduce neuron dependencies and memorization of data. 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).

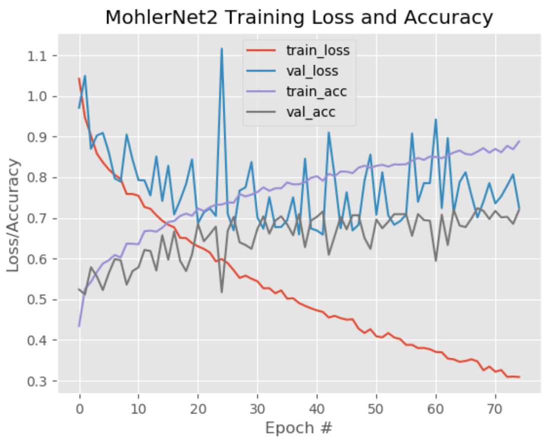

Figure 2: MohlerNet2 Accuracy, Learning Rate:0.01

	Precision	Recall	F1	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/Tot	0.72	0.72	0.72	750

Table 2: MohlerNet2 Classification Results

Once the first trials of the SGD optimized results were obtained we observed the effect of modifying the optimizer for the network. While using the AdaGrad and Adam optimizers, 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. In order to attempt to increase the accuracy of the network we must take in to account that we are dealing with a relatively small data set (by modern standards). With the small number of images being used to train the network there is a high probability that we will experience over-fitting while attempting to increase performance. In order to counter this, we next applied data augmentation to the training set. This allows us to modify the images in moderate ways (rotations, flips, sheering, etc.) such that they are presented to the network in a new way each epoch. Included with this approach was the shift to presentation of the data in batches. a wide variation of batch sizes were tested, anywhere from 16 to 96. Through testing it is heuristically determined that batches of 16 patterns tended to offer better results. The inclusion of data augmentation yielded another (relatively) considerable increase in accuracy and F1-Score. As shown in Table 3, both metrics increased by 4% such that the MohlerNet2 implementation could classify with 76% accuracy. However, from Figure 3 it can be seen that from the gap between training accuracy and validation accuracy, the network is experiencing some mild over fitting, which indicates that some modifications may be able to improve the performance further, these remain to be discovered.

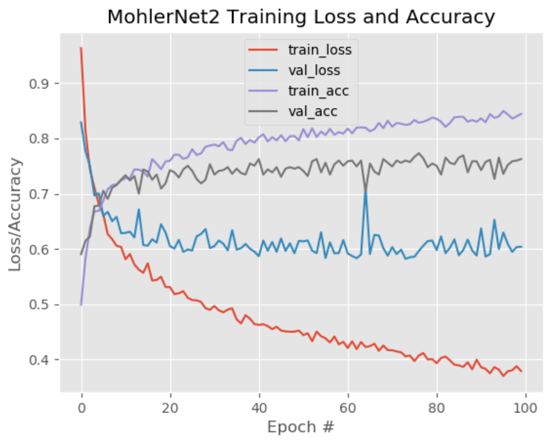

Figure 3: MohlerNet2 with Data Augmentation Accuracy, AdaGrad Optimized

	Precision	Recall	F1	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/Tot	0.76	0.76	0.76	750

Table 3: MohlerNet2 with Data Augmentation Results

2.3 MohlerNet3

The next generation implementation of MohlerNet (MohlerNet3) is based upon a simplified version of the layer architecture applied by Alex-Net [2]. This iteration has a large increase in layers and number of computations relative to its predecessor. MohlerNet3 employs a total of five convolution layers, the first two layers yield 64 and 128 activation maps respectively, and each layer is paired with its own 2×2 max-pooling operation. The following convolution layers use 64, 32 and 32 activation maps respectively with a single 2×2 max-pooling at their completion. From here we flatten and regularize the data with a dropout operation with a probability of 50% and apply the standard softmax activation to recieve the output class probabilities. A summary of this network topology is shown below.

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

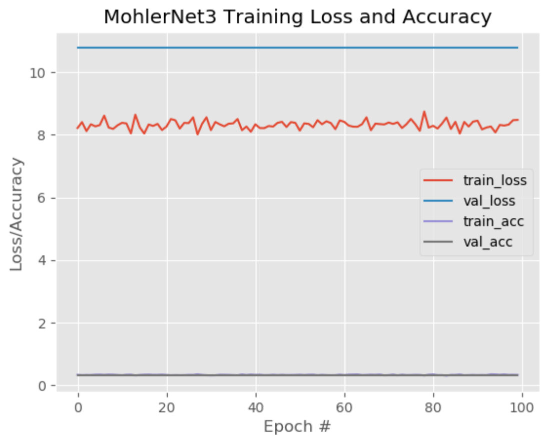

Figure 4: MohlerNet3 with LR=0.2

In initial experimentation with the learning rate we encountered the effect of a rate which is too large for the network to cope with. As shown in Figure 4, with a learning rate of $\eta = 0.2$ the network diverges. This results in the network accuracy plummeting to 33% which, with three classes is the equivalent of randomly selecting a class. In this case no learning occurs and the propagation of the error through the network in order to correct for the gradient fails. As found with the previous iteration, when the network is trained without the application of data augmentation there is a loss of performance. MohlerNet3 obtained an accuracy of 71%, and as such its complete results have not been included here. However, when the data is augmented the accuracy of the network is boosted to a value of 77%, however, the F1 and recall scores fell slightly, which indicates this network is more subject to false positives and false negatives.

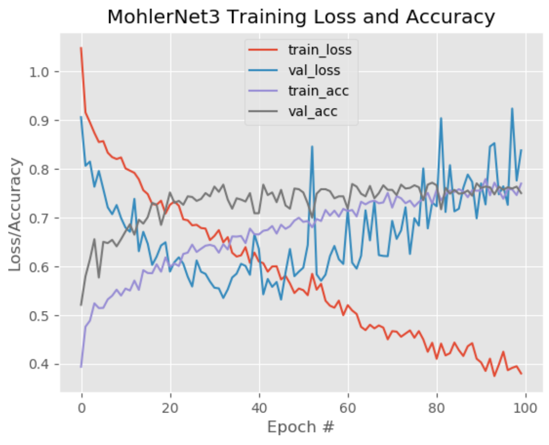

Figure 5: MohlerNet3 with Data Augmentation Accuracy, AdaMax Optimized

	Precision	Recall	F1	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/Tot	0.77	0.75	0.75	750

Table 4: MohlerNet3 with Data Augmentation Results

2.4 MohlerNet4

Applying what has been discovered in the first two editions of MohlerNet, we modify the architecture of the network into a medium sized network (relative to the first two) which balances a low number of layers with the aim of the higher accuracy provided by the larger of the two networks. From this we select the AdaMax optimizer once more and operate under the assumption that the augmentation of the data consistently provides a boost in performance. Using the architecture shown below, we use 32 activation maps in the first two convolution layers with 3×3 kernels and 2×2 max-pooling, and 64 activation maps in the final convolution layer. However, in this iteration we add an additional fully connected layer prior to the output layer with 64 fully connected neurons.

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

Using the ReLU activation for all layers we achieve results that are comparable to the much larger MohlerNet3 as shown in Table 5.

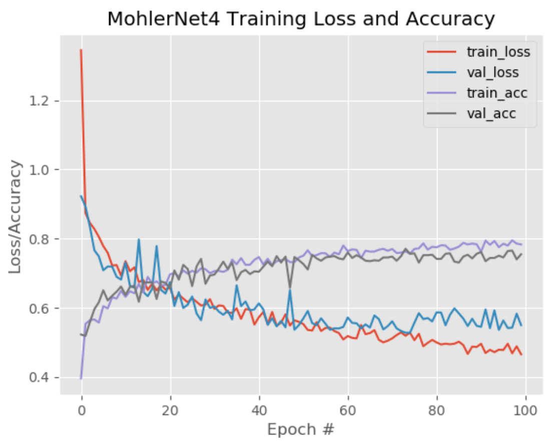

Figure 6: MohlerNet4 with Data Augmentation Accuracy, AdaMax Optimized

	Precision	Recall	F1	Support
Cat	0.76	0.61	0.67	262
Dog	0.64	0.79	0.71	249
Panda	0.93	0.92	0.92	239
Avg/Tot	0.77	0.77	0.76	750

Table 5: MohlerNet4 with Data Augmentation Results

With the promising results delivered by MohlerNet4, we next experimented with the modification of the activation function in the first dense layer. Instead of the use of ReLU as is done in all previous networks and layers, we attempted the use of the hyperbolic tangent activation function. In terms of classification accuracy, this yielded the best network. Edging out the previous best by a single percentage point, we were able to obtain a classification percentage of 78%. While this result was able to be replicated to within a single percent, it should be noted that the loss and accuracy characteristics from Figure 7 are not typical. It can be seen that the validation accuracy out strips the training accuracy by nearly 20%, as well as displaying the training loss not approaching the desired value of zero within the trained period.

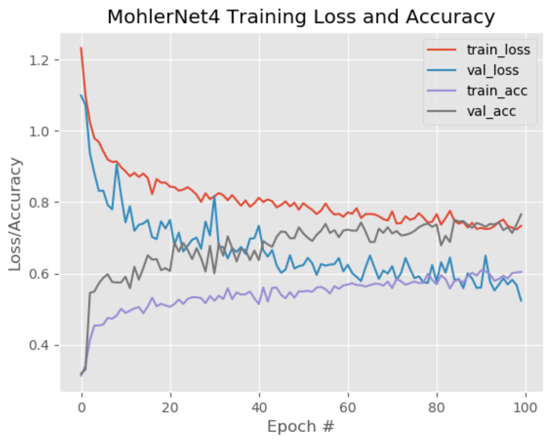

Figure 7: MohlerNet4 with Data Augmentation Accuracy, AdaMax Optimized

	Precision	Recall	F1	Support
Cat	0.83	0.51	0.63	262
Dog	0.64	0.83	0.72	249
Panda	0.85	0.95	0.9	239
Avg/Tot	0.78	0.76	0.75	750

Table 6: MohlerNet4 with Data Augmentation (tanh variant) Results

2.5 Network Comparisons

When comparing the final performances of all three network implementations several factors should be considered, here we will include the raw accuracy, F1 scores, and computational burdens. At first pass we can see that each iteration of MohlerNet was able to increase in accuracy, even though it was only for a total of 2%. In terms of F1 scores the networks were all within a single point from each other, however the difference between the accuracy and F1 score is greatest for MohlerNet4. Lastly, we can look to the number of learn-able weight parameters in the network. As would be expected, the network with the greatest number of layers and applied convolution filters (MohlerNet3) carries the greatest computational burden. This makes it apparent that the expansion of a network to be larger in terms of layers and neurons does not necessarily equate to an increase in task performance. As in all practical engineering tasks there are trade offs to be made in selecting the best network. In terms of raw accuracy it is apparent that MohlerNet4 is the clear winner, however it is 138% larger than MohlerNet2 in terms of necessary parameters, this directly translates to an increase in computational and memory burden. For a potential implementation it would be necessary to justify a decision for the increased burden for the single percentage of accuracy that is gained from selecting version four over version two. Conversely, it could be said that of the tested networks MohlerNet3 is the worst implementation. While it does provide the mean accuracy it is tied for the lowest F1 score metric and requires nearly double the parameters to accomplish its classification process. These figures are summarized in Table 7 below.

Network	Accuracy (%)	F1 Score	# of Weights
MohlerNet2	76	0.76	68,547
MohlerNet3	77	0.75	178,691
MohlerNet4	78	0.75	94,435

Table 7: Network Metrics

3 Conclusions

In this project we have demonstrated the ability to construct a number of functional convolution neural networks with the ability to classify images of cats, dogs, and pandas. While the classification is far from perfect, in its best instance MohlerNet4 was able to classify at an accuracy of 78% which is a 12% increase upon the original ShallowNet CNN that was provided. Due to the small data set provided for training and testing we have shown that data augmentation is an essential method used to boost the accuracy of the network (by as much as 5% in the tested cases). While each optimizer applied was able to create a convergent network, the AdaMax optimzer proved to offer the greatest performance for each of the three networks, and the Adam optimizer consistently provided a reduction in performance compared to the other optimization schemes. Through multiple iteration and critical parameter tuning, we have demonstrated that a convolution neural network can be successfully taught to recognize and categorize images of animals and from this we have displayed essential pieces of analysis and the necessary design steps to create a functional network.

A Code Listings

Listing 1: Top Level CNN Implementation

```
# USAGE
# python shallownet_animals.py --dataset ../datasets/animals

# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from pyimagesearch.preprocessing import ImageToArrayPreprocessor
from pyimagesearch.preprocessing import SimplePreprocessor
from keras.preprocessing.image import ImageDataGenerator as IDG
from pyimagesearch.datasets import SimpleDatasetLoader
from networks.nn.MohlerNet import MohlerNet1, MohlerNet2  ,
   [+]MohlerNet3, MohlerNet4
from keras.optimizers import SGD
from keras.optimizers import Adagrad, Adam, Adamax
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import csv
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ["CUDA_VISIBLE_DEVICES"]="-1"

def SaveResults(filename,results):
    if os.path.isfile(filename):
        with open(filename,'a', newline='') as resultFile:
            writer = csv.writer(resultFile,delimiter=',')
            for line in results:
                writer.writerow(line)
    else:
        with open(filename,'w', newline='') as resultFile:
            writer = csv.writer(resultFile,delimiter=',')
            for line in results:
                writer.writerow(line)

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
```

```
args = vars(ap.parse_args())

# grab the list of images that we'll be describing
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the image preprocessors
sp = SimplePreprocessor(32, 32)
iap = ImageToArrayPreprocessor()

# load the dataset from disk then scale the raw pixel intensities
# to the range [0, 1]
sdl = SimpleDatasetLoader(preprocessors=[sp, iap])
(data, labels) = sdl.load(imagePaths, verbose=500)
data = data.astype("float") / 255.0

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.25, random_state=42) #Keep random at 42 for
           [+]consistent evaluations

# convert the labels from integers to vectors
trainY = LabelBinarizer().fit_transform(trainY)
testY = LabelBinarizer().fit_transform(testY)

# initialize the optimizer and model
print("[INFO] compiling model...")
#opt = SGD(lr=0.01)

dataAugmentation = True
LR,EPS = 0.01, 0.1
#opt = SGD(lr=LR,momentum=0.9)
#print("Learning Rate: ",LR)#,"\tEpsilon: ",EPS)
#opt = Adagrad(lr=LR,epsilon=EPS) #LR should be 0.01 and eps 0.1
   [+]for this optimizer
#opt = Adam()
opt = Adamax(lr=LR)
print("Network Parameters:\n",opt.get_config())
model = MohlerNet4.build(width=32,height=32,depth=3,classes=3)
model.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the network
print("[INFO] training network...")
```

```
numEpochs = 50

batch_size = 16

if dataAugmentation == True:
    train_datagen = IDG(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip=True)

    train_datagen.fit(trainX)
    H = model.fit_generator(train_datagen.flow(trainX,trainY,
       [+]batch_size=batch_size),
                   steps_per_epoch=(2250//batch_size)
                  ,epochs=numEpochs,
                  validation_data=(testX,testY))
else:
    H = model.fit(trainX, trainY, validation_data=(testX, testY),
            batch_size=batch_size, epochs=numEpochs, verbose=1)

 #evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=batch_size)
results = classification_report(testY.argmax(axis=1),
        predictions.argmax(axis=1),
        target_names=["cat", "dog", "panda"])
print("\n",results)

#Format data to be written to CSV file
#-----------------------------------------------#
test = results.split()
for i in range(len(test)):
    test[i].replace('/','')
test[19:22] = [''.join(test[19:22])]
test = list(filter(None, test)) # fastest
test.insert(0,'')
test = np.reshape(test,(5,5))
NetDict = {
    1 : "ShallowNet",
    2 : "MohlerNet2",
    3 : "MohlerNet3",
    4 : "MohlerNet4"
    }

OptDict = {
    1:"SGD",
    2:"AdaGrad",
    3:"Adam" ,
    4:"Adamax"
    }
Network = NetDict[4]
Optimizer = OptDict[4]

filename = Network+"_opt-"+Optimizer+"tests"
fcsv = filename+".csv"
SaveResults(fcsv,test) #write results to file
print("Results saved as: ",filename)
#END RESULTS STORAGE
#------------------------------------------------------------------#
   [+]

# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure(filename)
plt.plot(np.arange(0, numEpochs), H.history["loss"], label="
   [+]train_loss")
plt.plot(np.arange(0, numEpochs), H.history["val_loss"], label="
   [+]val_loss")
plt.plot(np.arange(0, numEpochs), H.history["acc"], label="
   [+]train_acc")
plt.plot(np.arange(0, numEpochs), H.history["val_acc"], label="
   [+]val_acc")
plt.title(Network+ " Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(filename)
plt.show()
```

Listing 2: 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.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.