MSDS 458 Deep Learning
Juan Campos
Assignment 2/3
8/11/2019

1. Executive Summary

This research was conducted to obtain practical experience and an understanding of designing, training, and evaluating a Convolutional Neural Network (CNN). Using a dataset of cat and dog images, the study's main goal was to build an image classification CNN that distinguishes between cats and dogs. Of particular interest is understanding which features and credit assignments are used by the CNN to learn and to classify the images.

2. Data Preparation, Exploration, Visualization

The dataset contains 3,000 images of cats and dogs. The images were obtained from the Kaggle Dogs vs. Cats Competition. The original dataset contains 25,000 images. Due to hardware constraints, the dataset was reduced down to 3,000 images for this study. The smaller dataset would alleviate training time and memory time-outs. The images are in color and 280 x 300 in size. Each image is labeled with the word cat or dog and a sequential number appendix (i.e. cat.01, dog.4094). The animals are displayed in varying positions on the images. Some images show the full body of the animals while others show only a headshot. Different breeds and a myriad of colors of the cats and dogs are represented in the images (Figure 1).

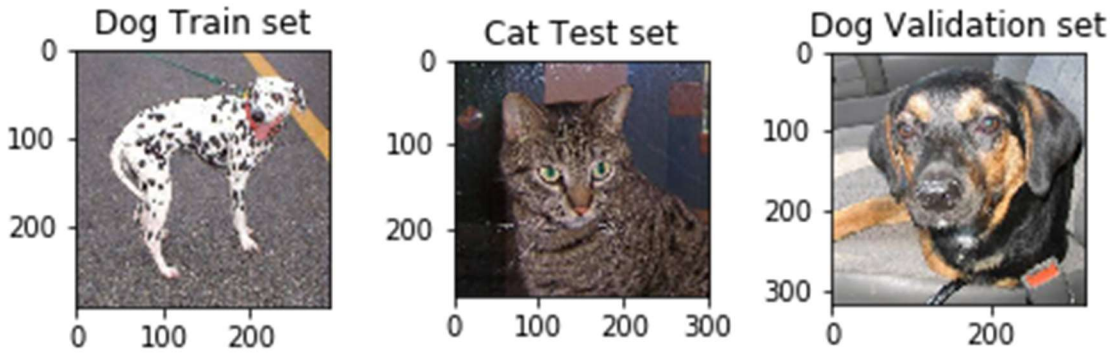

Figure 1: Example of dataset images

Most of the images in the dataset also contain noise in the form of objects in the environment. For instance, blankets with designs, cages, leashes, human arms and hands, and even human faces appear in the images (Figure 2). The noise

in the images, the small dataset, and the varying sizes and orientation of the animals in the images may make it more difficult for the CNN algorithm to distinguish between cats and dogs.

Figure 2: Example of noise in the images

To improve the algorithm's ability to generalize the images, the CNN algorithm was programmed to randomly flip, zoom, or angle the training set images.

The dataset was split into three sub-datasets for training of the CNN and testing and validating its classification ability. The split consists of 2,000 Train images, 500 Test images, and 500 Validate images. The two classification categories are represented 50/50 in each of the sub-datasets. The labels of the images were set as factory type with the values: 'cat' or 'dog'.

3. Implementation and Modeling

The Python code and the Jupyter Notebook environment were used to conduct the study. Several Python packages were used to conduct the study with the packages Keras and the Tensorflow backend being employed to perform the Deep Learning tasks such as training the CNN.

A modified version of the Visual Geometry Group -16 (VGG-16) CNN architecture was selected for the study. This network architecture uses an input size of 224 x 224. The input size of the images used in the study was downsized to 64 x 64 to satisfy computer demands. Other parameters that are distinctive of the VGG-16 architecture are that it has a 3 x 3 receptive field; a convolution stride of 1; padding of 1 for the receptive field of 3 x 3; five max pooling layers of 2 x 2 with a 2 pixel stride, five convolution layers followed by two fully connected layers (first connected layer has

4096 channels and the second has 1,000 channels, one for each class), the activation function of ReLU, and the final layer is a Softmax classification layer. All hidden layers are equipped with ReLU.

For this study, the first convolutional layer starts small with a width of 64 channels (3 x 3 x 64). Each subsequent convolutional layer increases by a factor of two after each max-pooling layer until 512 reached. To lessen the model's overfitting the data and help improve its accuracy, regularization was added to each convolutional layer. To force all of the nodes to learn a portion of the input, a dropout rate of .50 was applied to each convolutional layer block. In addition, early stopping was applied to the model with a patience value of 10. This helps prevent the model from over-fitting. If the loss function does not decrease after 10 epochs, the model will stop learning and will move on. Ideally, this will help save training time as well. However, due to the computer limitations, the maximum number of epochs that was ran was five. Consequently, this feature did not provide a benefit to the model. Finally, the learning rate was reduced each time that the accuracy of the model did not improve by 2. The VGG-16 CNN architecture using in this study is shown in Appendix A.

Several runs of the model were conducted using 3 or 5 epochs. The runs each took approximately one hour to complete. Performance metrics were calculated and graphed. Finally, SHAP was employed to explain the output of the CNN.

4. Results

The model's classification accuracy score settled between the range .50 and .52 for each run. The area under the ROC curve was also in the range of .50 and .52 after each run, indicating that the algorithm was not fitting the data well. Figure 3 displays the Accuracy history for two runs, one implementing three epochs and no Regularization and the second using five epochs and Regularization.

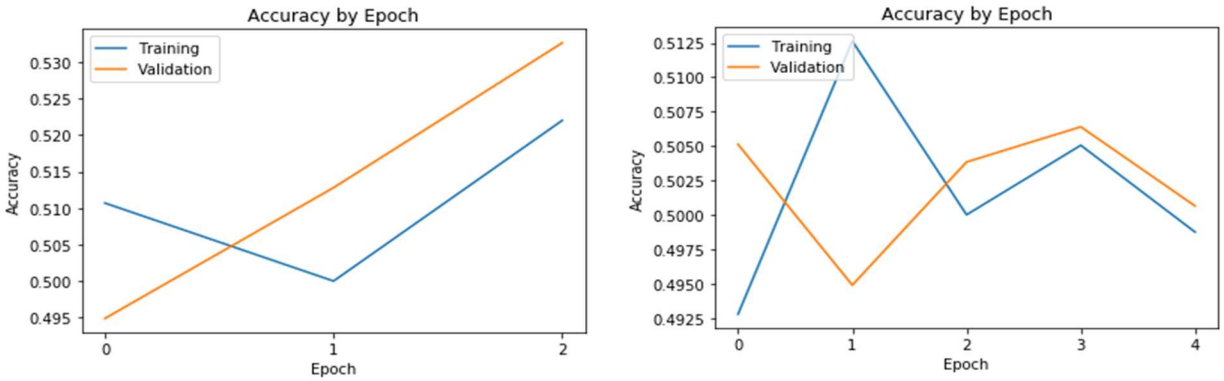

Figure 3: Example accuracy history for three epochs run and five epoch runs.

Figure 3 shows that the training accuracy took a dip from .51 to .50 by the second epoch in the three-epoch run and began to increase to .52 by the last epoch. The classification accuracy of the validation set increased steadily with each epoch. In the five-epoch run, the model's accuracy increased from .4925 to .5125 during the second epoch but lowered to .50 by the third epoch. The accuracy continued to rise and fall by small increments during the final two epochs ending with .52 accuracy. The validation accuracy fell from .5050 to .4950 during the second epoch but increased to .5125 during the third epoch. As with the training dataset, the classification of the validation dataset decreased and increased during the final two epochs achieving a .5005 accuracy score.

Figure 4 displays the area under the ROC curve for the runs referenced above. The graph on the left represents the three-epoch run and the graph on the right represents the five-epoch run. As be seen, the generated area under the ROC curve scores were .515 for the three-epoch and .50 for the five-epoch runs.

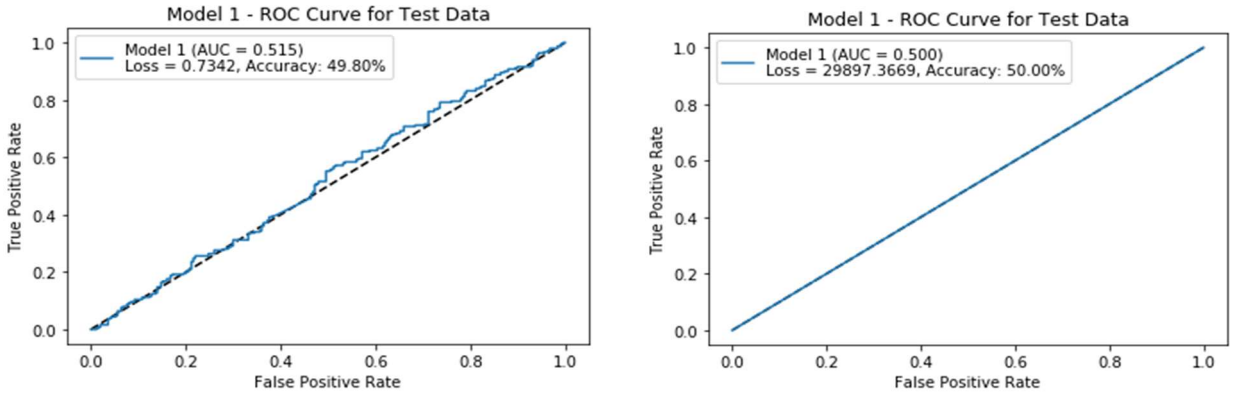

Figure 4: Area under the ROC curve for three-epoch and five epoch runs.

CNN's are black-box models. It is difficult to obtain an understanding of what they learned due to the many deep layers. This is especially true of the deepest layers. Although these layers learn the higher-level features of the images, they are very far removed from the images. To help understand the CNN output, the Shapely values or SHAP was employed to explain the results. Figure 5 and Figure 6 shows the SHAP values and graphs that help explain how the model classified the images. The first four images on the left represent cat false negative classifications. The four images on the right represent dog true positive classifications. What is peculiar is that a dog image appears in the cat false negatives and a cat appears in the dog true positives. It was expected that cat false negatives would only retrieve cat images and dog true positives would only retrieve images of dogs.

At this time, it is unclear if this is the result of the .50 accuracy of the classification algorithm, some other issue with the algorithm, or if some of the images are misclassified.

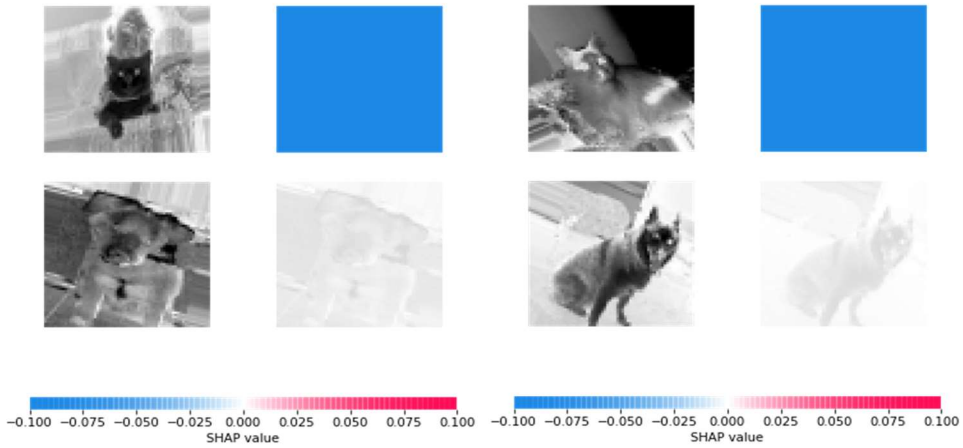

Figure 5: SHAP Value Graphs: 5 Epoch with Regularization Run

Figure 5 represents the results of one of the runs using five epochs and regularization. The first four images on the left represent the cat false negatives and the four images on the right represent the dog true positive. The images in the first and third columns are input images themselves. The images in the second and fourth columns display the SHAP analytics. The SHAP output suggests that the model was unable to detect/extract features from the images to help it distinguish between the two animals. The first two images in the second and fourth columns are completely highlighted in blue. This suggests that the CNN filters were unable able to 'see' any features in the image nor to isolate the animal from the background. Although it appears that the CNN filters were able to isolate the animals from the background for the bottom images, the images do not show any red coloring, indicating that the model was unable to extract features or assign credit to the images to classify them.

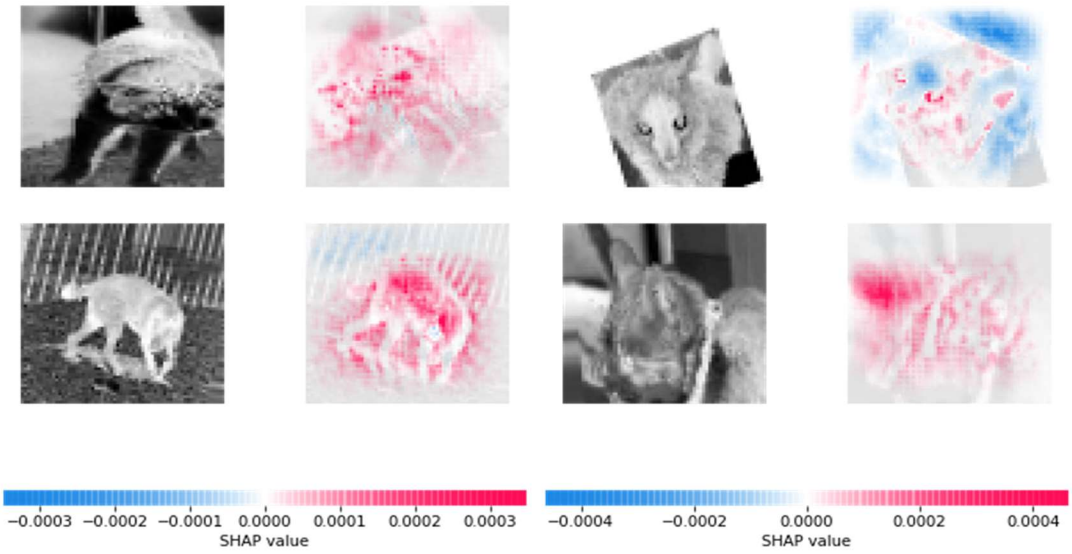

Figure 6: SHAP Graphs: 5 Epoch Run with No Regularization

Figure 6 represents the results of one of the runs using five epochs but no regularization. For some reason the CNN was able to extract features and assign credit to the images when regularization was not applied to the CNN. As can be seen certain pixels are highlighted in red. These areas the features that drove the CNN classifications. The areas highlighted in blue appear to be what the CCN is considering background. However, for the bottom image of the third column, it appears that part of the background was considered part of the animal as it is highlighted in red.

In examining the first convolutional layer, it appears that the CNN filters are beginning to detect edges and separate the elements in the images. As can be seen in Figure 7: the cat is apparent. Being able to isolate the animal is a good first step for the model to learn and correctly classify the images.

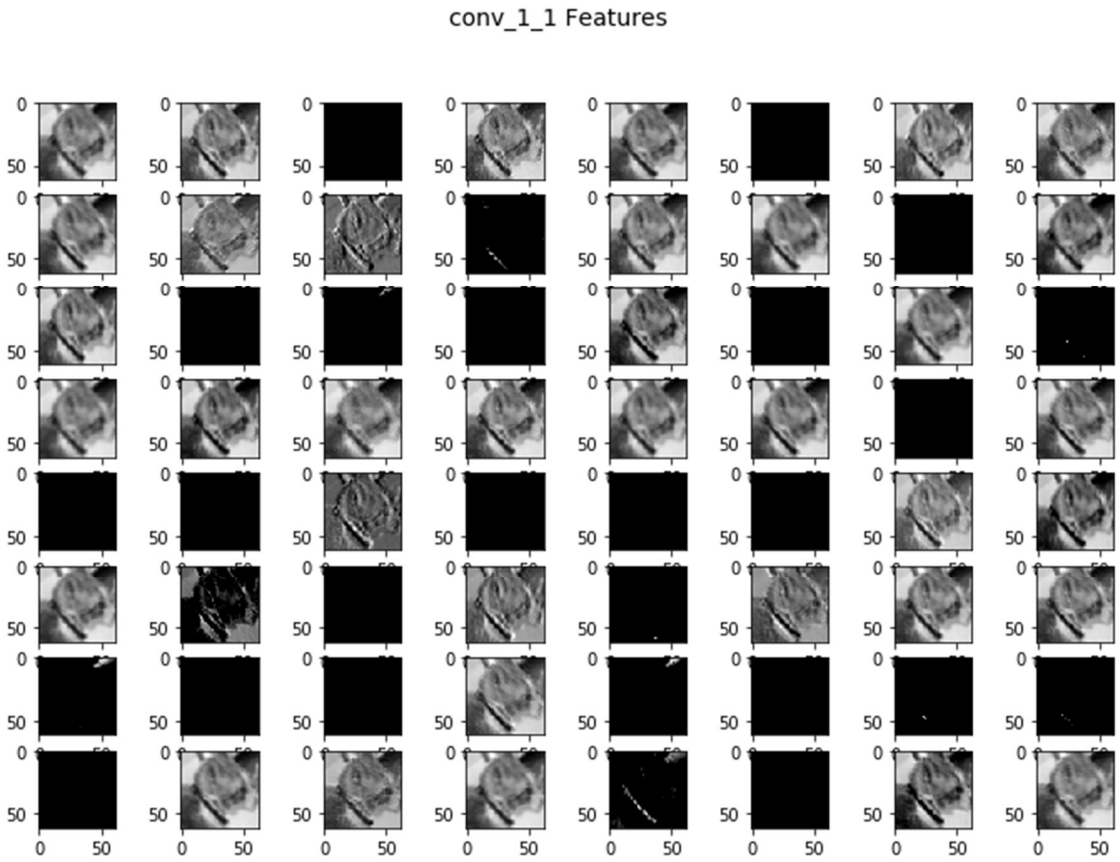

Figure 7: Convolution Layer 1 Features: Run 5 epochs with Regularization

5. Conclusion and Recommendations

In this study, the CNN had difficulty classifying the two classes and was not able to distinguish between a cat or a dog more effectively than random guessing. This might be attributed to the low number of epochs in each run and the small dimension of the dataset. Neural Networks, especially CNNs, perform best with large datasets and given sufficient time to learn. Future runs of the network should allow for larger dimension dataset and higher epoch rates. Another factor that could have adversely impacted the learning and accuracy rates are the images themselves. The images are very noisy and the network had difficulty isolating the animals from their

backgrounds. Reducing the image size to 64 x 64 may have also impacted the model accuracy as the resizing resulted in lost pixels, or information for the CNN. Larger images should be used in later runs of the network. Other questions arose during the examination as well highlighting the need for the images to be inspected to ensure that they are classified correctly. Any misclassified images may also confuse the CNN. In addition, it appears that the CNN was unable to extract features from the images when regularization was applied. Additional research is needed to explain why regularization would have such an impact on the model. Finally, although the CNN appears to be able to learn to isolate the animal from the other elements in the image, it appears that this ability is lost in later and deeper layers of the network.

Note: Rough Plan for Final Project listed in Appendix B.

Appendix A

VGG-16 CNN

```
def create_vgg16(x=None):
        model = Sequential()


  # Conv Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(image_height, image_width, 1), name='conv_1_1',
                     activity_regularizer = l1(0.001)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_1_2',
                    activity_regularizer = l1(0.001)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.50, name = 'dropout1'))


# Conv Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_2_1',
                    activity_regularizer = l1(0.001)))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_2_2',
                    activity_regularizer = l1(0.001)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Dropout(0.50, name = 'dropout2'))


    # Conv Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same', name='conv_3_1',
```

```
                activity_regularizer = l1(0.001)))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', name='conv_3_2',
                activity_regularizer = l1(0.001)))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', name='conv_3_3',
                activity_regularizer = l1(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.50, name = 'dropout3'))

# Conv Block 4
model.add(Conv2D(512, (3, 3), activation='relu', padding='same', name='conv_4_1',
                activity_regularizer = l1(0.001)))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same', name='conv_4_2',
                activity_regularizer = l1(0.001)))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same', name='conv_4_3',
                activity_regularizer = l1(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.50, name = 'dropout4'))

# Conv Block 5
model.add(Conv2D(512, (3, 3), activation='relu', padding='same', name='conv_5_1',
                activity_regularizer = l1(0.001)))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same',name='conv_5_2',
                activity_regularizer = l1(0.001)))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same',name='conv_5_3',
```

```
                activity_regularizer = l1(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.50, name = 'dropout5'))


# FC layers
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes


# optimize and compile the network architecture
opt = keras.optimizers.RMSprop(lr=learn_rate, rho=0.9, epsilon=None, decay=0.0)
model.compile(loss=keras.losses.binary_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])


# return the constructed network architecture
return model
```

Appendix B

Tentative Rough Plan for Final Project

Problem

Develop an image classifier CNN to automatically analyze breast cancer histology slide images for cancer risk.

Data Preparation, Exploration, Visualization

Images will be preprocessed by rescaling them and will most likely will remain in RGB color unless computer performance becomes an issue (then the images will be transformed to gray-scale). Graphs and table will be constructed where appropriate and sample images will be examined for insight.

Research and Design and Implementation

A VGG-16 CNN or other CNN will be used to differentiate between positive and negative cancer images. Python, Jupyter Notebook, and the Keras and Tensorflow packages will be used to code the algorithm. SHAP will be used to explain the CNN output.

Results and Evaluation of Models

The algorithm will be evaluated using Accuracy Score, AUROC, and a confusion matrix (specificity and sensitivity)

Conclusion and Management Recommendation

What was learned in the study will be discussed along with any pertinent recommendations for management.