Introduction: Summary and problem definition for management

Assignment 7 builds on the work done in assignment 6 in which we developed a deep neural network (a 'DNN') to identify handwritten digits in the MNIST dataset. In that problem we had 10 potential outputs – numbers ranging from 0 to 9. The DNN model used multiple layers of 'fully connected' neurons to classify which images represented each digit. We demonstrated that with 5 layers of neurons we were able to predict MNIST numbers with 97% accuracy.

This week, the problem is comparatively simple. It is a binary question – is this an image of a cat or a dog? In practice, however, the same DNN models do not perform as well on this task – they cannot differentiate the two classes given the input complexity. To improve their performance, we apply a technique known as 'convolution' – or filtering.

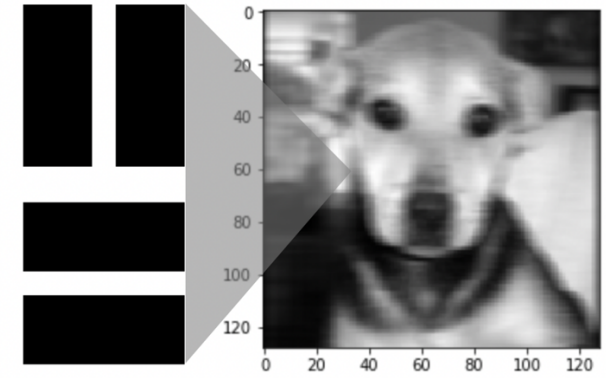
Assignment 7 – CNN models to classify images

Is this an image of a cat or a dog?

Research Design, Measurement & Statistical Methods

The data set for this assignment is 2,000 color pictures of dogs and cats and is more complex than the linear images of handwritten digits. To prepare the data for analysis, we need to turn each picture into a Numpy array which we can easily concatenate, reshape and manipulate. The complexity of each array will be a factor of image size - 128 pixels square vs. 64- and the number of layers or channels – color has 3 layers vs. 1 for greyscale images. These decisions will have an impact on processing time, so the objective is to minimize complexity while maximizing differentiation for classification.

```
fmap = np.zeros(shape=(7, 7, 1, 2), dtype=np.float32)
fmap[:, 3, 0, 0] = 1
fmap[3, :, 0, 1] = 1
plot_image(fmap[:, :, 0, 0])
plt.show()
plot_image(fmap[:, :, 0, 1])
plt.show()
```



An example of how a 'convolutional' filter is applied to each 4x4 pixel group – enhancing vertical & horizontal changes

Once the data is ready, we take the following steps:

1) Scale the data and split it into training & test data sets.
2) Test two models using assignment 6 'DNN' techniques.
3) Apply 'CNN 'method for filtering and processing images.
4) Compare the model timing and accuracy.

How 'convolution' works and why it helps:

In the reading for this assignment, the research points to the fact that in 'real' neural networks there is a filtering process prior to the data being input to the brain.[1] To simulate this natural process, we will apply a technique to filter the image and enhance specific vertical or horizontal attributes. This will help the neural network soften the inputs prior to applying a DNN similar to the one used on the MNIST data set. The technique is referred to as a 'Convolutional Neural Network' (CNN). Although this step adds complexity to the model, we will see that the CNN model performs much better than simply using a DNN.

Model overview:

Below is a summary of the models that were created in the process of this analysis and their performance comparison. Variables include the hidden neural network composition as well as the timing and accuracy scores of training & test sets. In the next section, I will review the build for each model and a visual comparison of each models' 'confusion' matrix.

	Model 1 – 128 x 128 Greyscale (5 layer DNN – Min-Max Scaler)	**Model 2 - 128 x 128 Greyscale (5 layer DNN – Robust Scaler)**	**Model 3 – 64 x 64 Color (CNN filter passed to DNN)**
Model Parameters & Performance	DNNClassifier hidden_units=[2000,1500,1000,500] Accuracy (train/test) .99/.58	DNNClassifier hidden_units=[2000,1500,1000,500] Accuracy (train/test) .1/.56	3 layer pass 'tf.learn.conv_2d' 2x pooling & 'fully connected' Accuracy = .94. Loss = .57
	Time to build: **~15 min**	Time to build: **~23 min**	Time to build: **~18 min**

[1] Géron, A. 2017. *Hands-On Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* Sebastopol, Calif.: O'Reilly. Source code available at https://github.com/ageron/handson-ml

Programming overview:

Below is the code that was used to prepare the data, develop & test the models, and a snapshot of the accuracy outputs.

Data preparation:
Each file from cat & dog is re-sized – for Model 1 & 2 128x128 greyscale (layer = 1); Model 3 = 64x64 color (layer = 3)

Example – 128 x 128 greyscale (layer = 1))

```
cats_1000_128_128_1 = np.zeros((1000, 128, 128, 1))
for ifile in range(len(cat_file_names)):
image_file_path =
            os.path.join('cats_dogs_images/cats',
            cat_file_names[ifile])
image = parse_grayscale_and_resize
            (image_file_path, size = (128, 128))
cats_1000_128_128_1[ifile,:,:,0] = image
```

Example – 64 x 64 color (layer = 3))

```
cats_1000_64_64_3 = np.zeros((1000, 64, 64, 3))
for ifile in range(len(cat_file_names)):
image_file_path =
            os.path.join('cats_dogs_images/cats',
            cat_file_names[ifile])
image = parse_grayscale_and_resize
            (image_file_path, size = (64, 64))
            cats_1000_64_64_3[ifile,:,:,:] = image
```

Model 1 (Min-Max Scaler) & Moel 2 (Robust Scaler)

```
# 5 layer model with reduced neurons for each layer: [2000,1500,1000,500] & output
with tf.name_scope("dnn"):
  hidden1 = neuron_layer(X, n_hidden1, name="hidden1", activation=tf.nn.relu)
  hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2",  activation=tf.nn.relu)
  hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3", activation=tf.nn.relu)
  hidden4 = neuron_layer(hidden3, n_hidden4, name="hidden4",  activation=tf.nn.relu)
  logits = neuron_layer(hidden2, n_outputs, name="outputs")
```

…. Accuracy Model 1 (train/test) = .99/.58. Accuracy Model 2 (train/test) = 1/.56

Model 3 (tflearn.DNN with embedded Convolution & pooling layers x3 w/ dropout) [2]

```
# Network = preprocessed image data - Input is a 32x32 image with 3 color channels (red, green and blue)
# 1: Convolution layer with 32 filters, each 3x3x3 > #2: Max Pooling layer
conv_1 = conv_2d(network, 32, 3, activation='relu', name='conv_1') > network = max_pool_2d(conv_1, 2)
# 3: Convolution layer with 64 filters > #3 Convolution layer with 64 filters
conv_2 = conv_2d(network, 64, 3, activation='relu', name='conv_2') > conv_3 = conv_2d(conv_2, 64, 3, activation='relu'..)
# 5: Max pooling layer > # 6: Fully-connected 512 node layer
network = max_pool_2d(conv_3, 2) > network = fully_connected(network, 512, activation='relu')
# 7: Dropout layer to combat overfitting > # 8: Fully-connected layer with two outputs
network = dropout(network, 0.5) > network = fully_connected(network, 2, activation='softmax')
# Wrap the network in a model object
model = tflearn.DNN(network, checkpoint_path='model_cat_dog_6.tflearn', max_checkpoints = 3,
            tensorboard_verbose = 3, tensorboard_dir='tmp/tflearn_logs/').
```

Accuracy = .93

Conclusion: The recommendation is to use model 3 – the convolution model (CNN). Although it takes a lot more memory to process the model, the accuracy of the test predictions is much higher than the other models . In the end, the additional processing complexity is worth the outcome.

Note: A shared version of the python notebook used to create this analysis can be found at Google CoLab:

https://colab.research.google.com/drive/1ihd88__vcV52NdAXZaNGWT0Pq11PIL9W

[2]Code source for model 3:
http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks