Hot Dog
Or
Not Hot Dog

JinZhao Su and Theodore Kim

Abstract:

Let us ask ourselves this physiological question. What is a Hot-Dog? Is it just a stick of meat between two buns? Throughout time and history, many have debated what exactly makes a Hot-Dog a Hot-Dog. Many creative minds have come close to an abstract definition of what a Hot-Dog is. Many philosophers have debated that a Hot-Dog is so much more than that; It's a message. To settle the debate on whether or not a Hot-Dog is actually a Hot-Dog or not, scholar JinZhao Su and Theodore Kim created many Neural network to mimic the brain in order to understand which features make a Hot-Dog, truely a Hot-Dog.

Approach:

To truly understand what makes a Hot-Dog a Hot-Dog, we have selectively chosen a dataset that has two categories. The photos in category one are what we can agree upon as a 100% Hot-Dog. The photos in category two are what we either consider 100%, not Hot-Dog or a questionable/fake Hot-Dog. The two categories were then loaded into a list:

```
|: import os

hot_dog_image_dir = 'train/hot_dog/'
hot_dog_paths = [''.join(hot_dog_image_dir+filename) for filename in
                            os.listdir(hot_dog_image_dir)]

not_hot_dog_image_dir = 'train/not_hot_dog/'
not_hot_dog_paths = [''.join(not_hot_dog_image_dir+filename) for filename in
                            os.listdir(not_hot_dog_image_dir)]

image_paths_train = hot_dog_paths + not_hot_dog_paths
y_dog=np.ones((len(hot_dog_paths),1),dtype=int)
y_not=np.zeros((len(not_hot_dog_paths),1),dtype=int)
y_train=np.concatenate((y_dog, y_not), axis=0)
# print(image_paths_train)
# print(y_train)

#############################################################################################################
#############
hot_dog_image_dir_test = 'test/hot_dog/'
hot_dog_paths_test = [''.join(hot_dog_image_dir_test+filename) for filename in
                            os.listdir(hot_dog_image_dir_test)]

not_hot_dog_image_dir_test = 'test/not_hot_dog/'
not_hot_dog_paths_test = [''.join(not_hot_dog_image_dir_test+filename) for filename in
                            os.listdir(not_hot_dog_image_dir_test)]

image_paths_test = hot_dog_paths_test + not_hot_dog_paths_test
y_dog_test=np.ones((len(hot_dog_paths_test),1),dtype=int)
y_dog_not=np.zeros((len(not_hot_dog_paths_test),1),dtype=int)
y_test=np.concatenate((y_dog_test, y_dog_not), axis=0)
# print(image_paths_test)
# print(y_test)
# print(y_test.shape)
```

Each picture has exactly a given height and width pixel dimension, along with three layers that correspond to their respective red, blue and green color intensity. By utilizing the Keras.processing.image library, the data was split into three dimensions in the following order(width, height, depth):

```
: def loadpath(image_paths,xsize=16,ysize=16):
    X=[]
    for location in image_paths:
        img=load_img(location,target_size=(xsize,ysize))
        sx=img_to_array(img) #(512, 382, 3)
#         print(x.shape)
#         sx=sx.reshape((1,)+sx.shape) #(1, 512, 382, 3)
#         print(sx.shape)
#         print(sx)
        X.append(sx)
        input_shape=sx.shape
    return input_shape,np.array(X)
```

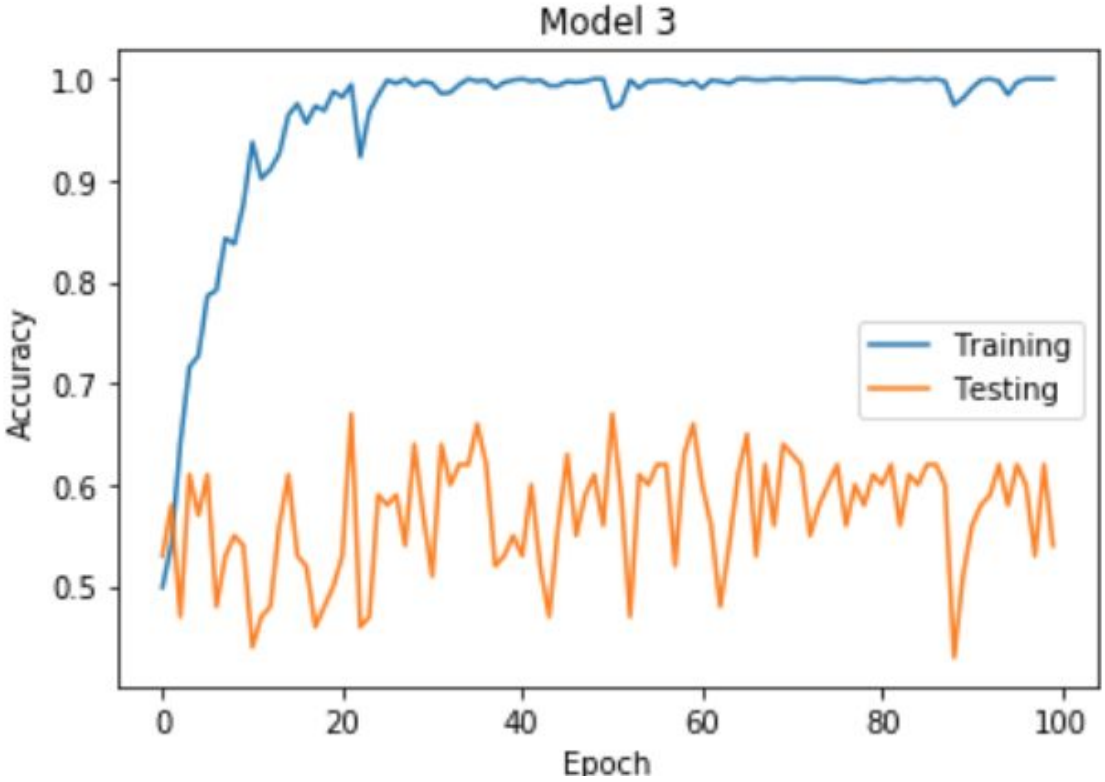
Thanks to the advances in science throughout the last few Centuries, there a library that generates entropy! By using the following code, the stimulating environment allows us to randomly split our datasets!

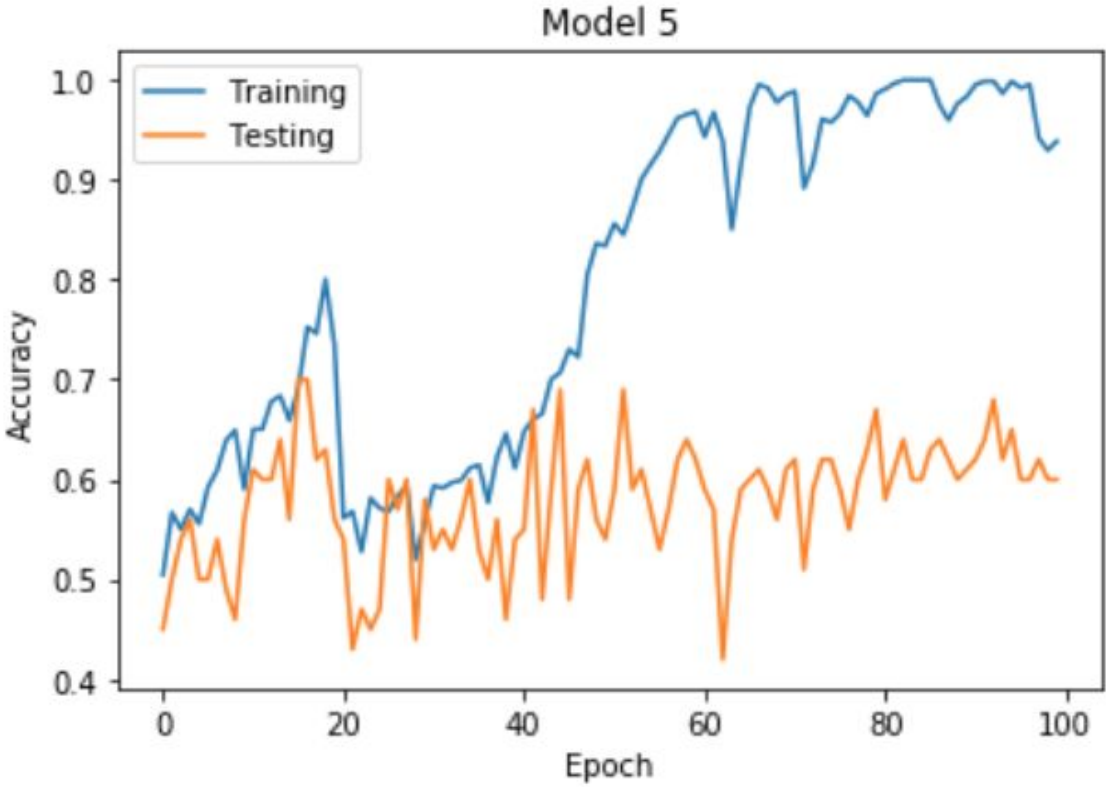
Models:

```
M1 = Sequential()

M1.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))

M1.add(Conv2D(64, (3, 3), activation='relu'))

M1.add(MaxPooling2D(pool_size=(3, 3)))

M1.add(Flatten())

M1.add(Dense(128, activation='relu'))

M1.add(Dense(1, activation='sigmoid'))

M1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 124, 124, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 41, 41, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 107584)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               13770880
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 13,790,401
Trainable params: 13,790,401
Non-trainable params: 0
_________________________________________________________________
```

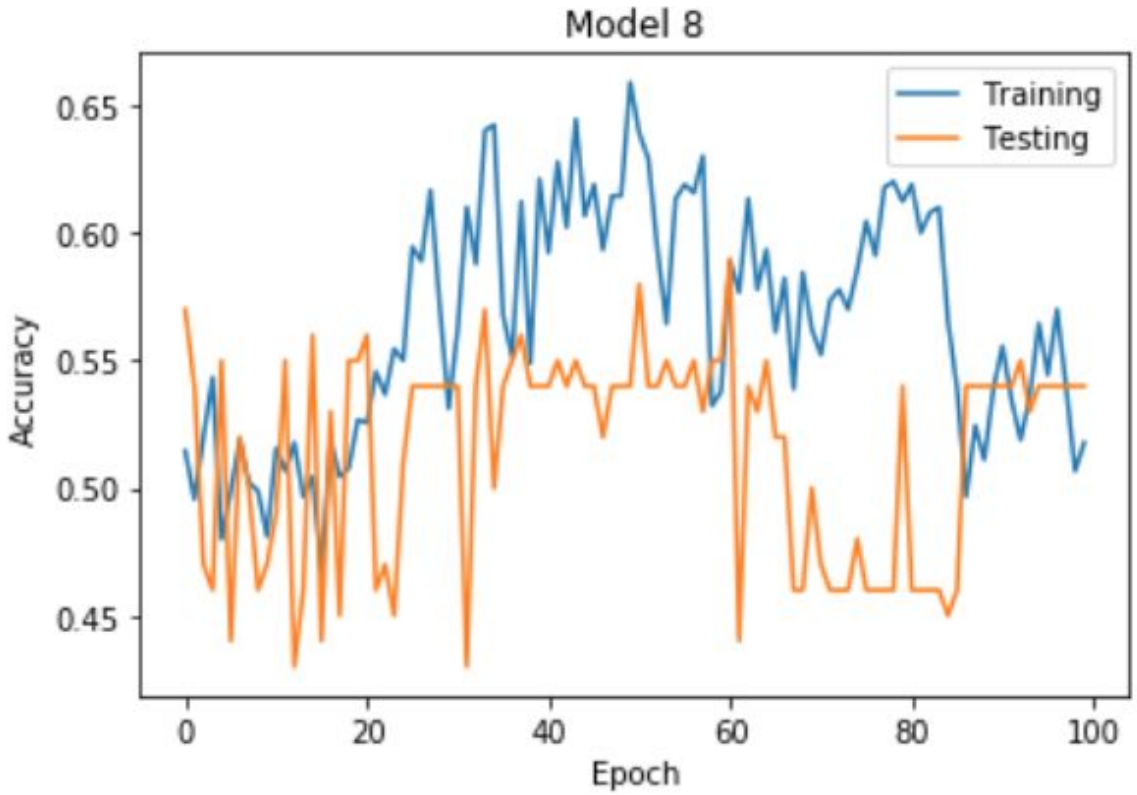
```
M3 = Sequential()

M3.add(Conv2D(32, kernel_size=(3, 3),strides=(1,1),activation='relu',input_shape=input_shape))
M3.add(Conv2D(64, (3, 3), activation='relu'))

M3.add(BatchNormalization())

M3.add(Dropout(.2))

M3.add(MaxPooling2D(pool_size=(3, 3),strides=(2,2)))

M3.add(Conv2D(64, (3, 3), activation='relu'))
M3.add(Conv2D(128, (3, 3), activation='relu'))

M3.add(Dropout(.2))

M3.add(BatchNormalization())

M3.add(MaxPooling2D(pool_size=(4, 4)))

M3.add(Conv2D(128, (4, 4), activation='relu'))
M3.add(Conv2D(128, (4, 4), activation='relu'))

M3.add(BatchNormalization())

M3.add(Dropout(.2))

M3.add(MaxPooling2D(pool_size=(4, 4)))

M3.add(Flatten())

M3.add(Dense(2100,activation='relu'))

M3.add(Dense(720,activation='relu'))

M3.add(Dense(1, activation='sigmoid'))

M3.summary()
```


Model 3
1.0
0.9
0.8
0.7
0.6
0.5
Accuracy
Training
Testing
0
20
40
60
80
100
Epoch


Model 5
Training
Testing
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100

```
M6 = Sequential()

M6.add(Conv2D(32, kernel_size=(2, 2),strides=(1,1),activation='relu',input_shape=input_shape))

M6.add(BatchNormalization())

M6.add(Conv2D(32, (2, 2),strides=(2, 2), activation='relu'))

M6.add(BatchNormalization())

M6.add(MaxPooling2D(pool_size=(3, 3),strides=(2,2)))

M6.add(Dropout(.2))

M6.add(Conv2D(64, (3, 3), activation='relu'))

M6.add(BatchNormalization())

M6.add(Conv2D(64, (3, 3), activation='relu'))

M6.add(BatchNormalization())

M6.add(MaxPooling2D(pool_size=(2, 2)))

M6.add(Dropout(.2))

M6.add(Conv2D(64, (4, 4), activation='relu'))

M6.add(BatchNormalization())

M6.add(Conv2D(64, (4, 4), activation='relu'))

M6.add(BatchNormalization())

M6.add(MaxPooling2D(pool_size=(2, 2)))

M6.add(BatchNormalization())

M6.add(Dropout(.2))

M6.add(Flatten())

M6.add(Dense(1600,activation='relu'))
M6.add(Dense(800,activation='relu'))
M6.add(BatchNormalization())
M6.add(Dense(400,activation='relu'))
M6.add(Dense(210,activation='relu'))

M6.add(BatchNormalization())

M6.add(Dense(1, activation='sigmoid'))

M6.summary()
```

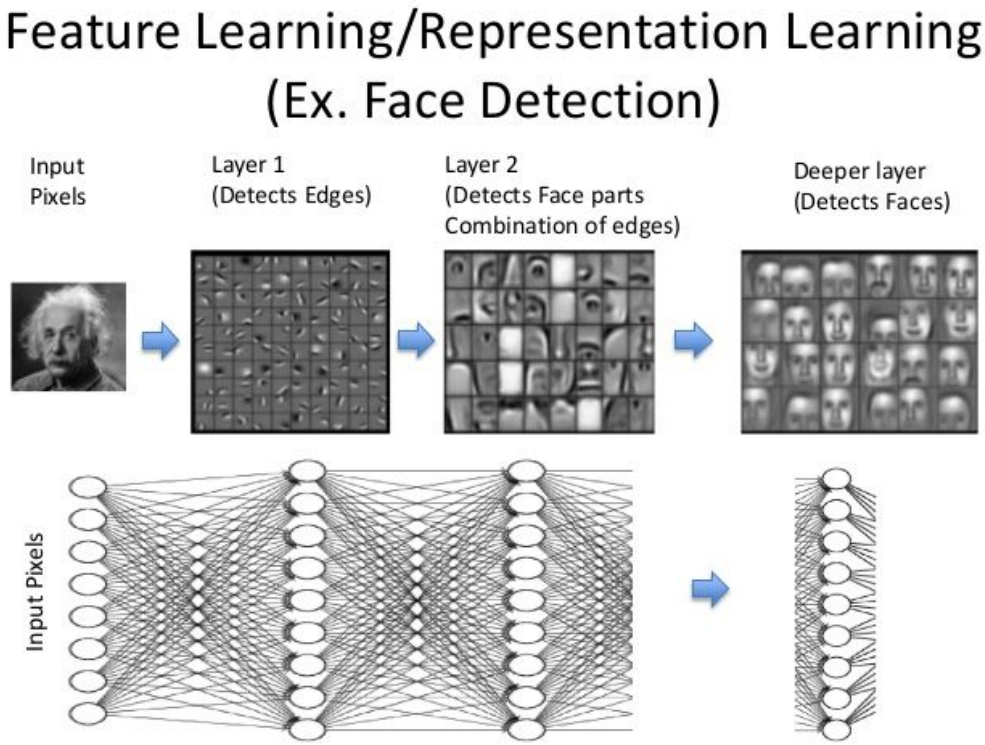
```
M8 = Sequential()

M8.add(Conv2D(32, kernel_size=(2, 2),strides=(1,1),activation='relu',input_shape=input_shape))

M8.add(BatchNormalization())

M8.add(Conv2D(32, (2, 2),strides=(2, 2), activation='relu'))

M8.add(BatchNormalization())

M8.add(MaxPooling2D(pool_size=(3, 3),strides=(2,2)))

M8.add(Dropout(.2))

M8.add(Conv2D(64, (3, 3), activation='relu'))

M8.add(BatchNormalization())

M8.add(Conv2D(64, (3, 3), activation='relu'))

M8.add(BatchNormalization())

M8.add(MaxPooling2D(pool_size=(2, 2)))

M8.add(Dropout(.2))

M8.add(Conv2D(64, (4, 4), activation='relu'))

M8.add(BatchNormalization())

M8.add(Conv2D(64, (4, 4), activation='relu'))

M8.add(BatchNormalization())

M8.add(MaxPooling2D(pool_size=(2, 2)))

M8.add(BatchNormalization())

M8.add(Dropout(.2))

M8.add(Flatten())

M8.add(Dense(1600,activation='sigmoid'))
M8.add(Dense(800,activation='sigmoid'))
M8.add(BatchNormalization())
M8.add(Dense(400,activation='sigmoid'))
M8.add(Dense(210,activation='sigmoid'))

M8.add(BatchNormalization())

M8.add(Dense(1, activation='sigmoid'))

M8.summary()
```


Model 8
Training
Testing
Accuracy
0.65
0.60
0.55
0.50
0.45
0
20
40
60
80
100
Epoch

The Models created were based on trial and error. We begin at the beginning of Model1. Model1 was a complete beauty, but it was also an accident. We needed to

have the loss function base on something that can separate the photos, but instead, we accidentally used mean square error. This renders, the data useless, but fear not. We will learn from our mistakes.

Model2 to Models8 were the the evolutionary steps that the neural networks evolved into. Model 2 began as an understanding of the non-linearity of the function relu. Relu is like the spaghetti of the glories day of the Roman Empire. The concept is insanely complex, but it's rewards are so delicious. Without going in-depth, Relu is non-linear. This is why when you call upon a Convolutional 2D, you need to use it twice. For an example, let us get into facial recognition:



If the data was linear, then the formula would be y=w1*w2*w3*x. But because of the principle of Relu, you will need the equation to be reshaped to be around y=w3*max(0,w2*max(0, max(w1*x))). The advantage of using multiple layers is that it learns different representation at each layer. For example, a network that detects a human in an image, will learn edges in the first layer, shapes in the next, body parts in the next and finally humans in the last. This is precisely Model 2 was based upon. The ideology that by using two Convolutional 2D networks each time, you can gain maybe the left curve of the tip of the Hot-Dog and then zoomed out to see the tip of the Hot-Dog. The rest of models were based upon Model 2 with some changes of layering, pixel intensity, but a few were an exception.

Few other layers that we used was Flatten, Dense, Dropout,BatchNormalization and Max pool. Flatten is needed to take the three-dimensional Matrix and output a single one-dimensional array. Dense is used to make hidden layouts that can compute the data; Such as output=activation(dot(input,kernel) +bias). This gets us to get a formula that can be best suited to determine the question; is it a Hot-Dog? The Dropout layer is used because it is to mimic the real-life situation. Not all data is the same, and thus what the drop-out layout would do is determine the feature that is the least important and drop them out. The BatchNormalization is used because it is a technique that provides any layer in a neural network with inputs that are zero mean/unit variance. This layer was added so that the neural network will function better. Finally there is Maxpooling. Maxpooling will get a nxn pixel range and get the max number of all the

numbers in the nxn range and get the highest number. This is used so that only the most intensive pixel will be used to calculate the pixel that actually matters.

Model here is what our glorious TA crafted. The reason why his neural network is tested is so that we have a standard of measurement. It's not to determine, whose network is better. If it was then surely he'll be at a disadvantage. However, according to our measurements, our neural network had an accuracy around the same of his neural network. We will talk about how performance is measured later on.

Model 7 is where a whole new concept was introduced. Based on only the sigmoid method introduced in Mnst demo lab, we decided to build a neural network that's only based on the sigmoid function. The results were astonishing because the accuracy of that model was exactly zero.

Model 8 to Model 9 was the evolutionary change that combined both the sigmoid and relu. The need for relu in Convolutional 2D and Max pool was effective, but diversity was missing from the neural network. That's when the idea of introducing the sigmoid function after the need of relu was introduced. Model 8 took advance that surprised scholar Jin. By using the sigmoid function after the necessity of relu for Maxpooling and Convolutional 2D, the model worked relatively better. But in theory, it was amazing. Just image, after figuring out all the features, you plug it into the sigmoid function and an boolean is erected. It's almost poetic.

Measurements of success:

The accuracy measurement that was used was binary_accuracy instead of the normal categlory_accuracy. Here's the difference:

Binary_accuracy:

K.mean(K.equal(y_true, K.round(y_pred)))

Binary accuracy is more like a softmax. Unlike a guaranteed yes or no, it gives us a rounded percentage.

Category_accuracy:

K.mean(K.equal(K.argmax(y_true, axis=-1), K.argmax(y_pred, axis=-1)))

Category gives us a 1 is it's true. It then gives an average of how many 1/total_tested.

Conclusion:

The model that was the most successful was a range between Model 6 and Model 8. However, the features that 100% defines what a Hot-Dog is,is still among the mist. The technology is close, but not there. The question still remains; What exactly makes a Hot-Dog a Hot-Dog?