Using Neural Networks to Identify Images of Hotdogs

JinZhao Su and Theodore Kim
New York University, js9202@nyu.edu, tk1931@nyu.edu

***Abstract* - Image processing and object identification is a popular application of machine learning, from Optical Character Recognition to facial recognition in biometric security applications. This paper outlines yet another application of computer vision and object identification: identification of hotdogs from non-hotdogs. This paper employs a variety of neural network models to optimize the recognition accuracy of the application as a means of achieving "perfect" recognition (an accuracy of 1.0 or close to 100% recognition).**

Keywords - Hotdogs, not hotdogs, machine learning, neural networks

FIGURE 1
Image of a hotdog

I. The Data□et

The dataset used in this study includes many labelled pictures of food. The two "types" of images utilized by the model, are picture specifically of hotdogs (or variants thereof, referred from here as simply, hotdogs) and those of food decidedly not hotdogs. The dataset was saved onto to the testing computer as a set of colored, JPEG encoded images. Each image was initially a different size and resolution, however, the images were normalized during the data preparation process.

Figures 1 and 2 are examples of the types of images included in the dataset. The dataset was found on Kaggle under the title, "Hot Dog Not Hot Dog" (https://www.kaggle.com/dansbecker/hot-dog-not-hot-dog).

FIGURE 2
Image of a "not hotdog"

II. Preparing the Data

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

```
# Get all the paths that have hotdog photos
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)]

#Get all paths of that have not-hotdog photos
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)]
   #Get all the paths combined into one
image_paths_train = hot_dog_paths +
   not_hot_dog_paths

y_dog = np.ones((len(hot_dog_paths),1),
   dtype=int)
#Get Y axis (all 1 when hotdog)
y_not = np.zeros((len(not_hot_dog_paths), 1),
   dtype=int)
#Get Y axis (all 0 when not hotdog)
y_train = np.concatenate((y_dog, y_not),
   axis=0)
#Get Y axis

# Get all the paths that have hotdog photos
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)]

#Get all paths of that have nor-hotdog photos
not_hot_dog_image_dir_test =
   'test/not_hot_dog/'
not_hot_dog_paths_test =
   [''.join(not_hot_dog_image_dir_test+filena
   me) for filename in
   os.listdir(not_hot_dog_image_dir_test)]
#Get all the paths combined into one
image_paths_test = hot_dog_paths_test +
   not_hot_dog_paths_test
#Get Y axis (all 1 when hotdog)
y_dog_test=np.ones((len(hot_dog_paths_test),1
   ),dtype=int)
#Get Y axis (all 0 when not hotdog)
y_dog_not=np.zeros((len(not_hot_dog_paths_tes
   t),1),dtype=int)
#Get Y axis
y_test=np.concatenate((y_dog_test,
   y_dog_not), axis=0)
```

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=[]
   #Loading images into
      (Sample_Size,Width,height,depth)
   for location in image_paths:
      img = load_img(location,target_size =
         (xsize,ysize))
      sx=img_to_array(img) #(512, 382, 3)
      X.append(sx)
      input_shape=sx.shape
   return input_shape,np.array(X)
```

Random entropy was simulated using the Scikit machine learning library's model selection function, and the test data set was divided randomly into a testing and training data set.

III. Approach

In order to create a refined version of the machine learning algorithm to achieve as close to perfect identification as possible, eight different neural network models were created in order to test how layering different types of networks and using different model activation algorithms would affect the effectiveness overall of the model. Below are the coded implementations of echo model as well as the summary of its layer and graph of its accuracies when applied to the training and test set.

Training and testing the models were done using the Python Anaconda development environment, Keras library, and Tensorflow environment on a PC equipped with a NVIDIA 1080TI graphics card (used to accelerate the computations)

```
# Basic model
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()
```

TABLE 1
Configuration Summary of Model 1

Layer (type)	Output Shape	Number of Parameters
Layer 1: 2D Convolutional	(None, 126, 126, 32)	896
Layer 2: 2D Convolutional	(None, 124, 124, 32)	18496
Layer 3: 2D Max Pooling	(None, 41, 41, 64)	0
Layer 4: Flatennd Network	(None, 107584)	0
Layer 5: Dense Network	(None, 128)	13770880
Layer 6: Dense Network	(None, 1)	129

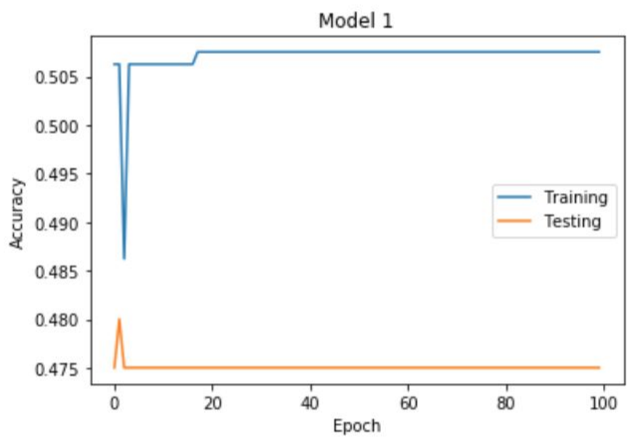


FIGURE 3
Training vs Testing Accuracy for Model 1

```
M2 = Sequential()
M2.add(Conv2D(32, kernel_size=(3,
   3),strides=(1,1),activation='relu',
   input_shape = input_shape))
M2.add(Conv2D(32, (3, 3), activation='relu'))
M2.add(MaxPooling2D(pool_size=(3, 3),
   strides=(2,2)))
M2.add(Conv2D(64, (3, 3), activation='relu'))
M2.add(Conv2D(64, (3, 3), activation='relu'))
M2.add(MaxPooling2D(pool_size=(4, 4)))
M2.add(Conv2D(128, (4, 4), activation =
   'relu'))
M2.add(Conv2D(128, (4, 4), activation =
   'relu'))
M2.add(MaxPooling2D(pool_size=(5, 5)))
M2.add(Flatten())
M2.add(Dense(2100,activation='relu'))
M2.add(Dense(1420,activation='relu'))
M2.add(Dense(128, activation='relu'))
M2.add(Dense(1, activation='sigmoid'))

M2.summary()
```

TABLE 2
Configuration Summary of Model 2

Layer (type)	Output Shape	Number of Parameters
Layer 1: 2D Convolutional	(None, 198, 198, 32)	896
Layer 2: 2D Convolutional	(None, 196, 196, 32)	9248
Layer 3: 2D Max Pooling	(None, 97, 97, 32)	0
Layer 4: 2D Convolutional	(None, 95, 95, 64))	18496
Layer 5: 2D Convolutional	(None, 93, 93, 64)	36928
Layer 6: Max Pooling	(None, 23, 23, 64)	0
Layer 7: 2D Convolutional	(None, 20, 20, 128)	131200
Layer 8: 2D Convolutional	(None 17, 17, 128)	262272
Layer 9: Max Pooling	(None, 3, 3, 128)	0
Layer 10: Flattened Network	(None, 1152)	0
Layer 11: Dense Network	(None, 2100)	2421300
Layer 12: Dense Network	(None, 1420)	2983420
Layer 13: Dense Network	(None, 128)	181888
Layer 14: Dense Network	(None, 1)	129

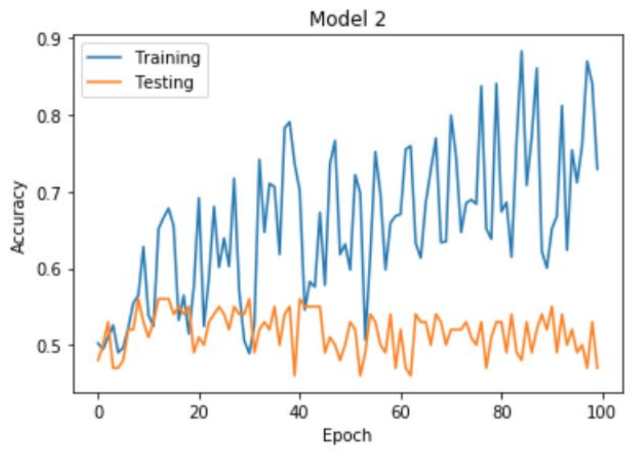


FIGURE 4
Training vs Testing Accuracy for Model 2

So far, the models perform increasing well for the testing set; however, they perform poorly when the model is extended to the test set (the models have poor variance). In an effort to improve the models' performance, Batch Normalization was added into the network as a means of both speeding up the network and improving its variance in other data sets.

To further reduce the variance of the models, drop out was introduced.

```
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()
```

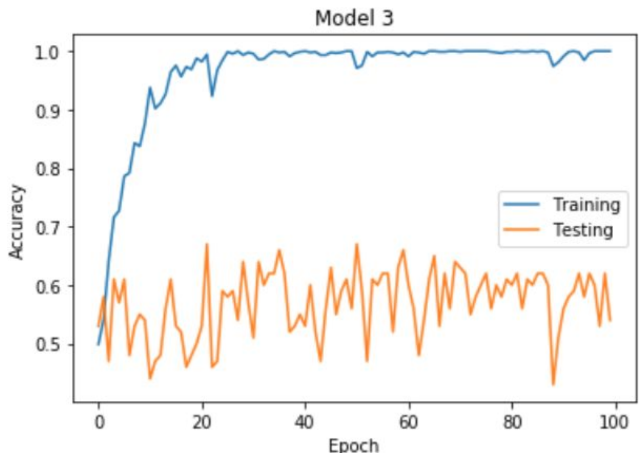


FIGURE 5
Training vs Testing Accuracy for Model 3

Model 3 demonstrates a lower variance than model 2 with the new measures taken to improve its average performance. However, the max accuracy reached by the model is still less than 70% and the average still around 50% (so the model performs no better than chance!). In Model 4, we add additional layers to try to achieve a more complex function and therefore, better performance.

```
M4 = Sequential()

M4.add(Conv2D(64, kernel_size=(3, 3),
    strides= (1,1), activation='relu',
    input_shape=input_shape))
M4.add(Conv2D(128, (2, 2),
    activation='relu'))
M4.add(MaxPooling2D(pool_size=(3,
    3),strides=(2,2)))
M4.add(BatchNormalization())
M4.add(Dropout(.420))
M4.add(Conv2D(64, (1, 1), activation='relu'))
M4.add(Conv2D(64, (2, 2), activation='relu'))
M4.add(MaxPooling2D(pool_size=(4, 4)))
M4.add(Dropout(.420))
M4.add(BatchNormalization())
```

```
M4.add(Conv2D(64, (3, 3), activation='relu'))
M4.add(Conv2D(128, (2, 2),
    activation='relu'))
M4.add(MaxPooling2D(pool_size=(3, 3)))
M4.add(BatchNormalization())
M4.add(Dropout(.420))
M4.add(Flatten())
M4.add(Dense(1600,activation='relu'))
M4.add(BatchNormalization())
M4.add(Dense(720,activation='relu'))
M4.add(BatchNormalization())
M4.add(Dense(1, activation='sigmoid'))

M4.summary()
```

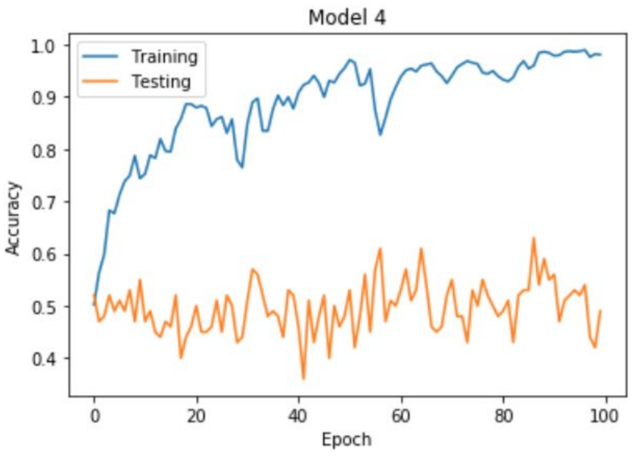
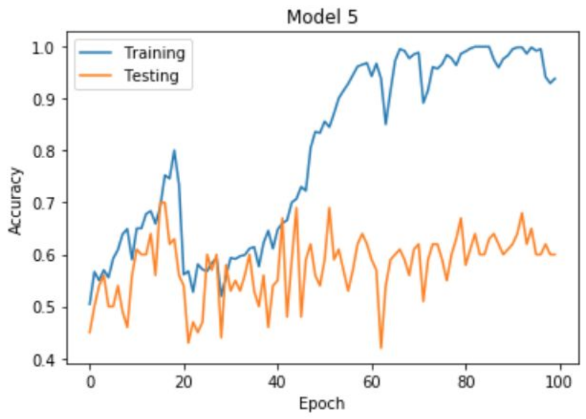
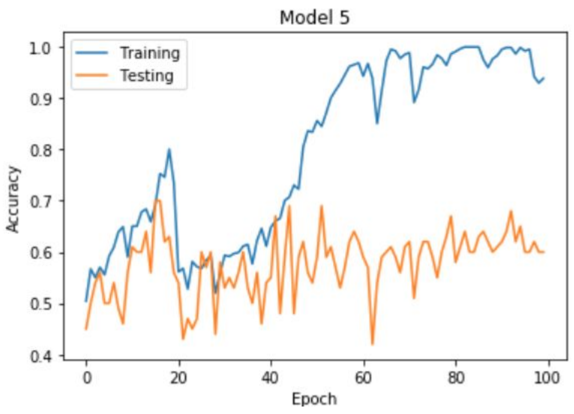


FIGURE 6
Training vs Testing Accuracy for Model 4

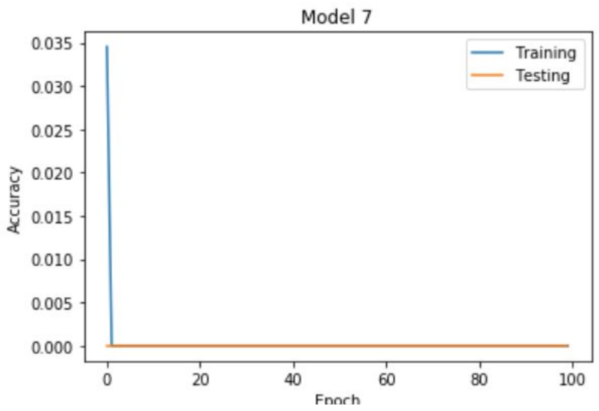
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), strides=(1, 1), activation='relu', input_shape=input_shap
e))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (4, 4), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(2000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(500, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()


#COMPARING OUR DATA WITH THE TA MODEL
```



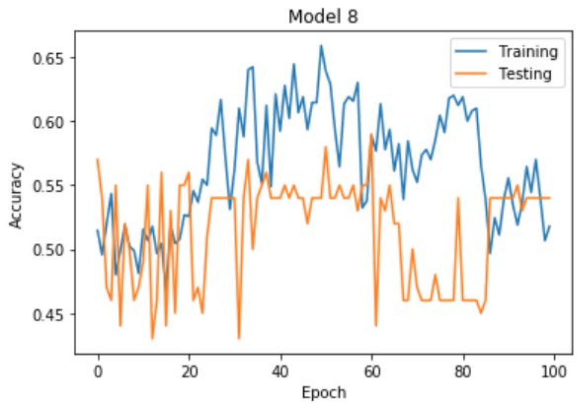
```
M6 = Sequential()

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()
```



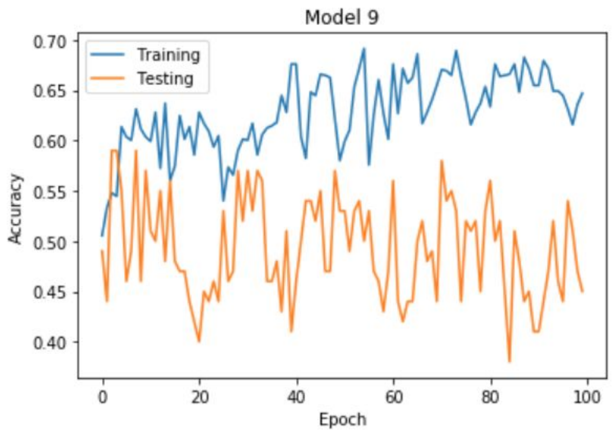
```
M7 = Sequential()
M7.add(Dense(1200, input_shape=input_shape, activation='sigmoid'))
M7.add(Dense(600, activation='sigmoid', name='hidden'))
M7.add(Flatten())
M7.add(Dense(1, activation='sigmoid', name='output'))
```



```
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()
```

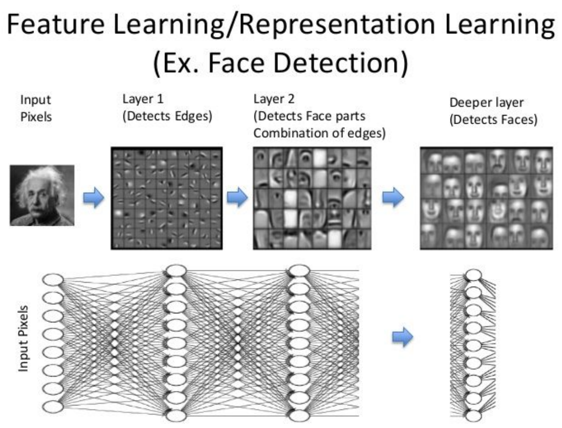


```
M9 = Sequential()

M9.add(Conv2D(32, kernel_size=(2, 2),strides=(1,1),activation='relu',input_shape=input_shape))

M9.add(BatchNormalization())

M9.add(Conv2D(32, (2, 2),strides=(2, 2), activation='relu'))

M9.add(BatchNormalization())

M9.add(MaxPooling2D(pool_size=(3, 3),strides=(2,2)))

M9.add(Dropout(.2))

M9.add(Flatten())

M9.add(Dense(1600,activation='sigmoid'))

M9.add(BatchNormalization())

M9.add(Dense(1, activation='sigmoid'))

M9.summary()
```



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.

Model 2 to Model s8 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:


Feature Learning/Representation Learning
(Ex. Face Detection)
Input Pixels
Layer 1 (Detects Edges)
Layer 2 (Detects Face parts Combination of edges)
Deeper layer (Detects Faces)
Input Pixels

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 hotdog and then zoomed out to see the tip of the hotdog. 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 hotdog? 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.

MEA□UREMENT□ O□ SUCCE□□

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.

CONCLU□ION

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