Using Convolutional Neural Networks for Detecting Hotdogs

Ian McKechnie[1]

[1]*Cis 4020, University of Guelph*
[1]imckechn@uoguelph.ca

***Abstract*— This paper can is a walk-through of how I would create Convolutional Neural Network (CNN) to classify Hot Dogs. Using TensorFlow, Keras, and some Sklearn methods I can create a binary CNN image classifier to identify if an image has a Hot Dog in it or not. I talk about how I would increase my training set size by performing transformations on the images in the dataset to create more images which would increase the accuracy of the classifier. Then I would normalize, center, and standardize the image pixel data so I can use them in a CN. Once the data is in the correct format for the Sequential Model, I could define a Keras early stopping variable that would help optimize the model to pick a correct number of Epochs so that the model is perfect balanced between underfitting and overfitting the data. Once the model is trained, I would then visualize the outputs and identify where the classifier is getting its predictions wrong using confusion matrices and printing the photos. I could then go back and tweak the model by hand to better optimize it for these errors. This will result in a CNN that classifies images as Hot Dogs or Not Hot Dogs**

***Index Terms*— Convolutional Neural Networks, Image Detection, Food Recognition, TensorFlow, Keras**

I. Introduction

The goal of this program is to classify images based on if they have hot dogs in them or not. This is based off a similar program in the hit HBO show Silicon Valley. This program will use a Convolutional Neural Network (CNN) to classify images. I chose CNNs for this project due to their widespread application in real-world large-scale image recognition projects such as Tesla's and Waymo's self-driving cars[1] and Google Lens[2]. The program will accept the already labelled images from the *Kaggel Hot Dog – Not Hot Dog*[3] image set. To do the image preprocessing and normalization to create the Neural Network Model. To create the actual model, I will be using Keras and TensorFlow. The reason for using Keras and TensorFlow is because TensorFlow has better documentation and has better visualization tools than Pytorch which is it's leading competitor which will allow easier bug fixing[4]. I am using Keras in this project because it offers the best tools for working with large, non-grey-scale images and since it integrates well with TensorFlow (It comes built in). This will allow them to work well together and allow faster development time.

II. Project Setup

Why I chose this topic for the assignment, why I chose CNNs, the dataset from *Kaggle*, and an in-depth reason on why this project would be using TensorFlow over Pytorch.

A. What Initially Drew Me to this Project Idea

Before I started this project, I knew wanted to create a Neural Network (NN) model. We did not get to develop a NN in class and I wanted to build one myself. Since Sklearn is not great for writing NNs[4] I needed a different library to do this project. I know large companies often use TensorFlow or Pytorch for creating large AI projects so I decided I would use one of them as the basis for this project. All I needed was relatively simple and approachable task that I could do. Luckily, I watch *Silicon Valley* and that gave me some ideas.

B. Why I Chose Hot Dog – Not Hot Dog

I am a big fan of the hit HBO show *Silicon Valley* and I remembered in some cis 4020 lectures, Professor Neil Bruce used *Hot Dog – Not Hot Dog* as examples of image classifier problems so I thought it would be a fun project to try. It seemed simple enough as I only needed a dataset of Hot Dog images.

C. *The Kaggle Hot Dog – Not Hot Dog Image Set*

Doing some quick searching around I found an already labeled and sorted image set offered by *Kaggle* for people who want to do a Hot Dog classifier. *Kaggle* is a website that has Jupyter Notebooks and datasets already organized for people to download and practice creating their own ML projects. The image set is organized perfect because the images are already divided in a testing and training set. And within those sets the data is also split into a Hot Dog and Not Hot Dog sets. So, there is no minimal organization that I'll need to do for this. The only issue with this dataset is it is small. There are only 500 training images. This is not a lot as for Neural Networks you want as much data as possible to train your model[5]. Also, how the image set is organized is not great for training a classifier. There are 250 training images, but also 250 testing images. I prefer to have a train-test-split of 80/20 split. I would have to move some of the testing data over to achieve this ratio. This is simple as I can just drag and drop the files over by hand. For this paper though I will not be doing that because I am using image transformations to make more images.

D. *The Environment I Used to Create the CNN*

I would use TensorFlow and write it in a Jupyter Notebook. The goal of this project is to demonstrate understanding for the concept and help visualize and label every step I take and why. For a lot of code walk throughs and demonstrations Jupyter Notebooks are used for this reason[6]. For example, a Jupyter Notebook is great for this because I can clearly label each step and why I have chosen it. When choosing the library to create the actual CNN it was between TensorFlow and Pytorch. Pytorch is newer and offers faster development time. TensorFlow is older and offers slower development time but better documentation and better visualization tools which help make debugging easier[4]. It would make the most sense to use Pytorch for the faster development time for this project but I want to be a data scientist and most projects in industry are written in TensorFlow so I decided I would rather have a challenge understanding this initially and maybe taking more time to do it but having more marketable skills at the end. It also has better visualization tools for debugging and since the goal of this project is to present the results that makes TensorFlow the obvious choice for this project.

III. Getting the Images

This section will show examples of code, and methods that I would use to create the program with explanation of why I would use it.

A. *Importing the Image Set into Python*

This project uses local files (Since I downloaded the image set), so I need to load them into the program. There are two ways of doing this. The first is with the Pillow, the Python Image Library (PIL)[7] or with Keras[8], which is written to work with TensorFlow. When you import images into the program with Keras using this method they are stored in a batch data set[9]. Keras and TensorFlow both work with batch sets so there will be no problem keeping our data in this format.

You can import images with it using the follow code:

```
#Import local the images into Python
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

train_batch_size = 250
img_height = 512
  img_width = 382

train_ds =
tf.keras.utisl.image_dataset_from_directory(
       "./directory/of/images",
       Image_size=(img_height, img_width),
       Batch_size=train_batch_size)[9]

train_ds = train_ds.cache().prefetch()[10]
```

This code defines the image parameters, then loads the images into a Keras dataset. Then we apply the cache and prefetch method to this dataset. We need to apply Cache and Prefetch to this dataset due to its size. If we don't use it the program could crash because the computer would not be able to keep all the data in the cache and trying to access it all to apply transformations would break it. With Cache and Prefetch if the computer cannot fit the entire dataset into its cache and TensorFlow wants to use

the cached data, TensorFlow knows not to since the data is incomplete. So, then it will reprocess the remaining elements[10]

B. Amalgamating the Training Data

The training data is from *Kaggle* and it comes in a file structure where the 250 Hot Dog images are in a separate directory from the 250 Not Hot Dog images. We need to amalgamate them into one batch set of images into two since learning models only accept one list of images as parameters. There also must be an array of labels so we can keep track of if the image was a Hot Dog or not. To amalgamate the two image sets you can use the concatenate method[11]. This will just stack the 250 Not Hot Dogs onto the Hot Dogs image set. Our label set can just be set so the first 250 labels are Hot Dog, and the last 250 are Not Hot Dog.

This process must also be repeated with the testing set which is also split between two directories.

C. Generating More Images from the Image Set

Since we only have 500 training images in the data set, we are using and CNNs are more accuratew with more images, I would increase the number of images available to the learning model by creating more based off the already available images. I remember in class, Dr. Neil Bruce said you can apply transformations on the images to make new images and add them to the dataset to better train our model. Some of the transformations we can do are rotations, shifts, flips, zooms, and brightness changes[12]. There is a Keras method for this fittingly called ImageDataGenerator which I will be using. The transformed images made through ImageDataGenerator can look drastically different than the original which is great because it will allow our CNN to make better guesses on our testing set.

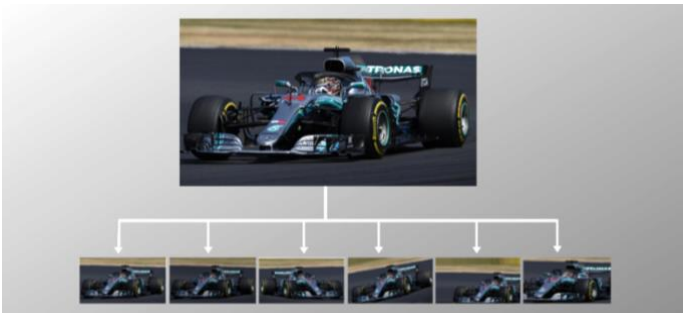


Figure 1, An example of what transformed images look like compared to the original[12]

As you can see from the above example image, the Mercedes F1 car images can look very different from the original with transformations performed on it.

I would randomly perform all five transformations on the images to create new ones. The first transform I would do is image rotations. You can randomly rotate images to any degree amount which maximizes the number of options that your images can become. It can be done with the following code:

```
#Rotating images using Keras ImageDataGenerator
Datagen = ImageDatagenerator(rotation_range = 10,
fill_mode='nearest')

newImage = datagen.flow(img, batch_size=1)
```

Next, I would perform shifts of the image. Shifting slides images in any direction on the x-axis or y-axis. This can allow images to be partially cut off on an axis or off center which would forces the CNN to adapt and learn to recognize hot dogs that aren't shown fully in the image. It can be done with the following code:

```
#Shifting images using Keras ImageDataGenerator
Datagen = ImageDataGenerator(width_shift_range=0.2,
height_shift_range = 0.2)

newImage = datagen.flow(img, batch_size=1)
```

I would also perform flips on the images. This is simply flipping the image in the y direction, x director, or both to create a new image. It can be done with the following code:

```
#Flipping images using Keras ImageDataGenerator
Datagen = ImageDataGenerator(horizontal_flip=True,
vertical_flip=True)

newImage = datagen.flow(img, batch_size=1)
```

Changing the brightness of images can also be useful in creating new images. Images are taken in all sorts of lighting conditions and with various cameras and camera sensors that have different lighting levels so building a CNN that is trained to handle this is important. To create a test set of images that are of different light levels you can use the following code:

```
#Flipping images using Keras ImageDataGenerator
Datagen = ImageDataGenerator(brightness_range = [0.4,
1.5, 2.6]) #These numbers are levels of brightness
that can be changed

newImage = datagen.flow(img, batch_size=1)
```

Finally, random zooming on the dataset is also a transformation that can be performed. The parameter to make ImageDataGenerator perform zooming is a single double which specifies how much the program will zoom in or out on the image(s) given to it. It is done with the following code:

```
#Zooming on the images using Keras ImageDataGenerator
Datagen = ImageDataGenerator(zoom_range=0.2)

newImage = datagen.flow(img, batch_size=1)
```

Doing all three of these transformations one by one is tedious and makes your program much longer than it needs to be due to a lot of repeating code. Luckily you can combine all these into one. It looks like the following:

```
Datagen = ImageDataGenerator(rotation_range = 10,
fill_mode='nearest', width_shift_range=0.2,
height_shift_range = 0.2, horizontal_flip=True,
vertical_flip=True, brightness_range = [0.4, 1.5,
2.6], img, batch_size=1)

newImage = datagen.flow(img,
batch_size=batch_size_of_hot_dogs)
```

D. Normalizing the Data

Image data needs to be normalized for Neural Networks to work on them. The standard is to normalize the pixel values, center the data, and standardize the images. Standardizing the data gives each pixel a range of one to zero, pixel centering scales the pixels to have a mean of zero. Finally, pixel standardization scales the pixels to have a variance of one[13]. This could be done manually but there are libraries to do this for you. The most used library is *Keras ImageDataGenerator*[14]. To scale the data, you use ImageDataGenerator and specify what range you want the pixels to be. It's done with the following:

```
#Scaling the images using Keras.ImageDataGenerator
Datagen = ImageDataGenerator(rescale = 1.0/255.0)

Train_iterator = Datagen(training_set_x,
training_set_y, batchSize = train_batch_size)
```

After the data is scaled between zero and one, we need to center the data. Luckily the ImageDataGenerator can also center the data for us. It is done by the following:

```
#Centering the image pixel data around zero using
#Keras.ImageDataGenerator
Datagen = ImageDataGenerator(featurewise_center =
True)

Train_iterator = Datagen(training_set_x,
training_set_y, batchSize = train_batch_size)
```

Finally, all we must do is standardize the pixel data in the images. Standardizing data shifts the distribution of the data so it will keep its mean of one, but also have a standard deviation of one which implies its variance is also one. This can also be done with the ImageDataGenerator from the Keras Library. It can be done with the following:

```
#Standardizing the image pixel data using
#Keras.ImageDataGenerator
Datagen =
ImageDataGenerator(featurewise_std_normalization =
True)

Train_iterator = Datagen(training_set_x,
training_set_y, batchSize = train_batch_size)
```

We can combine all three of these steps into one. It looks like the following:

```
#Normalizing, Centering, and Standardizing image
pixel data #with keras.ImageDataGenerator

Datagen = ImageDataGenerator(rescale = 1.0/255.0,
featurewise_center = True,
featurewise_std_normalization = True)

Train_iterator = Datagen(training_set_x,
training_set_y, batchSize = train_batch_size)
```

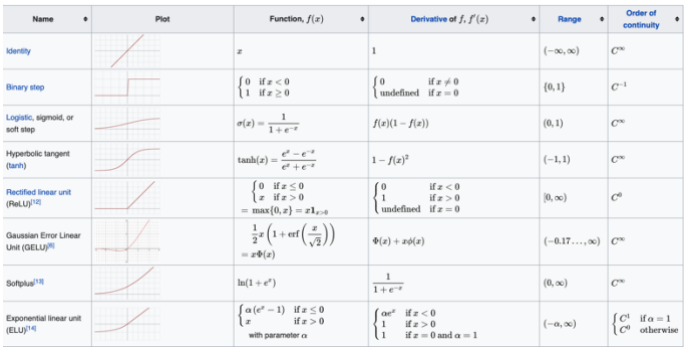
So now our images are ready for the Convolutional Neural Network and have not lost any information about them and could be reversed to their original formats if we needed.

IV. Training the Model with TensorFlow

This next section is creating the Convolutional Neural Network with TensorFlow and training it on the normalized image set.

A. Picking the Activation Function

We need an activation function for the CNN. There are many different activation functions to choose from[15] and the three most common functions are Sigmoid, Rectified Linear Activate Function (reLU), and Hyperbolic Tangent. I'll be using ReLU for the activation function for this project. This is because of its simplicity and reduced likelihood of vanishing gradient. This means the constant gradient in reLU has a faster learning rate.



Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$	$f(x)(1-f(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[12]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[6]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[13]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[14]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$

Figure 2 Examples of some activation functions you can choose for you model. Source: https://en.wikipedia.org/wiki/Activation_function

B. Creating the Convolutional Neural Network Structure

I will be using a sequential model for my CNN. The reason for this is sequential models work best when there is one input and one output. My model takes in a single image at a time and is a binary classifier so it will be perfect for this project.

We now get to choose the number of layers for the CNN. It must have a minimum of two layers, one for the input and one for the output layer. For the number of hidden layers, it depends on if the data is linearly separable. Since I don't know if it is or not that there are a few methods to check this[19]. I could try and cluster the images using and SVM or some other simpler model, or I could use a single perceptron to try and predict the data, and finally I could use a single hidden layer in my network to try and predict the data[20]. I will be using the single hidden layer to check this. This is because if the data turns out to not be linearly separable, I can just add in more hidden layers into my network to compensate for this and just retrain the network. So, to create the CNN network structure I can use the following code:

```
#Creating the CNN Structure
Model = keras.sequential([
layers.Dense(2, activation="relu", name="L1"),
layers.Dense(3, activation="relu", name="L2"),
layers.Dense(4, name="L3")
])[21]
```

And if the data was not linearly separable, I could add layers to it with the follow code:

```
#Adding layers to the model sequentially
Model.add(layers.Dense(5, activation="relu"))[22]
```

Or if it is overfitting, I could remove layers with the following:

```
#Removing layers from a CNN
Model.pop()
```

C. Training the Model

To train the model we must decide how many Epochs we want to use on the model. If we choose a number of Epochs that is too small, we get underfitting and if we choose too many, we get overfitting[23]. I can overcome this error by using a part of the training data to check the performance of the mode after each Epoch. I would monitor the loss and accuracy after each Epoch until I hit a desirable compromise between loss and accuracy that allows my training model to predict the optimal amount of Hot Dog images. Once a desired accuracy is met or a maximum loss value is met I can use *keras.callbacks.callbacks.EarlyStopping()* to stop the amount model from training more. You can initialize an earlyStopping variable as follows:

```
#An Creating an early stopping variable
earlyStopping
=callbacks.EarlyStopping(monitor="val_loss", mode =
"min", patience = 5, restore_best_weights = True)
```

The above code is set to watch for a validation loss, indicated with the parameter "val_loss". The

restore_best_weights parameter restores the best weights observed for the model. The mode parameter is set to 'min' when observing loss, and 'max' when observation accuracy. The patience parameter is the number of Epochs to run after the halt is reached. This is so the model can check if will improve after the halt is reached in case it finds a local max/minimum in the loss amount and could continue improving after that Epoch.

The earlyStopping variable is then passed in as a parameter into the models fit method as a parameter with the 'callbacks' tag as its key. From here we can run the training method on the data with a large number of Epochs to train our CNN. You can run the training with the following code:

```
#Training the Model on the Hot Dog Dataset
History = Model.fit(image_set,  label_set, batch_size = batch_size_of_all_images, Epochs = 50, validation_data = (val_images, val_labels), callbacks = [earlyStopping])
```

When you run this, it will a progress bar for each Epoch and include the val_loss and the val_accuracy each Epoch achieves. I have 50 Epochs set which is very large and I do not expect it to get to that level. That is because the earlyStopping will identify the optimal Epoch to stop at to prevent overfitting. This is most likely going to be at less than 20 Epochs[25].

V. Visualizing the Train

Although we got a numeric output of the accuracy and the loss at each Epoch, I still want a visual representation of the training over time, a Confusion Matrix, and to see which images it is classifying wrong to see if there is any pattern between them that I could correct for.

A. Visualizing the Loss Over Time/Epochs

We can create a simple matplotlib graph for the loss on the Y-axis and the Epoch number on the X-axis. We can get the history of it by using the history variable I set equal to the training method. An example of what I can expect the training loss to roughly look like is shown in Figure 3 bellow.

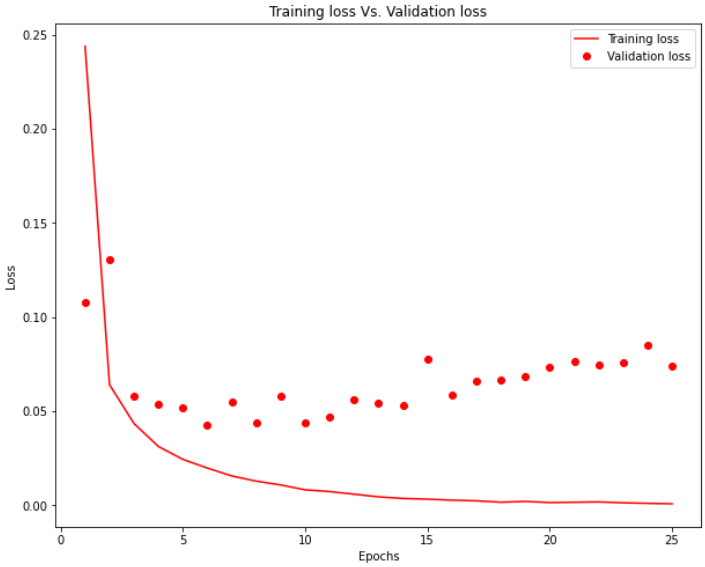


Figure 3, What expected loss per Epoch for the model should look like. Source: https://www.geeksforgeeks.org/choose-optimal-number-of-Epochs-to-train-a-neural-network-in-keras/

B. Confusion Matrix of the Hot Dog Binary Classifier

A Confusion Matrix is a great tool for easily visualizing how the classifier is predicting data correctly. If I was to use a simple sklearn.score[26] method it doesn't really show me how my image classifier is classifying images wrong, it just what percent of the testing data was predicted incorrectly. A Confusion Matrix is much better for this as I can see how many Not Hot Dogs are being labeled as Hot Dogs and how many Hot Dogs are being labeled as Not Hot Dogs. I should look like the one shown in figure 4 bellow.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	TN = 50	FP = 10
Actual: YES	FN = 5	TP = 100

Figure 4, A generic binary confusion matrix. Source: A. Visualizing the Loss Over Time/Epochs

C. *Printing the Mislabelled Images*

Another method I would want to use is just printing some of the mislabelled outputs. If I notice most of the errors were coming from false-positives I would print some of the false-positives to the console to see what they were. I can get the indexes of them from the Confusion Matrix method. If after printing I noticed for example that they were all Hot Dogs I would know that they were just mislabeled originally, and my model is most likely more accurate than I was led to believe. Or if my model was incorrectly predicting were groups of similar looking images, I would want to get more images that were similar to that group for training and see if that corrects my model. I will explain how that can be done in part IV.

VI. Rectifying Error Found in the Model

If the model has a large bias towards getting false-positives or false-negatives as seen in the Confusion Matrix above (Fig. 4) I will want to change my model to hopefully correct some of these and have the images properly labeled.

A. *Changing the Accuracy/Loss Amount at Training Time*

When I created the early stop variable, I set it to a specific loss level. I could change this to 80 percent accurate or 90 percent accurate instead (assuming the accuracy was already lower than these). This could make the training take longer but as a side effect would allow it to be better at guessing what the images are.

B. *Altering the Image set*

If I noticed that a certain image or group of similar images were failing my classifier, I could get more of them in my training set and use that to better train my classifier to predict them. For this I would want to use the image transform functions from section III.C to create more images from the original image and see if that fixes my classification error.

C. *Increase the Dropout of the CNN*

Dropout is when the CNN ignores certain neurons at random during the training phase and sometimes it increased the accuracy of a model. Neurons can be dropped out at any layer during training. I could try dropout with the following code:

```
#Adding Dropout to the CNN Model
Model.add(Dropout(0.2))
```

This is added between the layers when your first creating the model (section IV.B). The float parameter specified is the percentage of neurons that are being dropped and can be increased or decreased to improve the results[26].

D. *Add or Remove Layers from the CNN*

Adding or removing layers is also an option to increase the accuracy of the model. It is shown in section IV.B how to do this. Adding layers does not necessarily improve the accuracy of a model and decreasing can improve this. This is because if each feature of the data is already accounted for in a layer, any additional layers would just learn from a redundant feature which wouldn't improve our model and could even make it less accurate[27]. I would add in a layer see if that makes an improvement. If it worse than I would remove it. If it gets better than I would add another layer to see if that increases my accuracy further.

VII. Applications of this Software

I see two potential applications for a Hot Dog classifier in the real world. The CNN could be used to classify images of Hot Dogs for food apps where pictures of food is very common. The other option is to train retrain the model to identify pictures with genitals to warn users that the image might contain nudity.

A. *Food Identifier*

The most obvious application of this software is to identify pictures of Hot Dogs from users. Companies like Yelp, Uber Eats, Grubhub and more all have images of food on their apps and having software that could identify the images in menus from restaurants could be useful for customers. So,

this software could be open sourced and used by them to identify food photos on their app. It could also be used by social media companies where people post lots of pictures of their food. For example, if somebody posting an photo of their Hot Dog on Instagram, the app could recommend a Hot Dog hashtags for them.

B. Nudity Detection

Hot Dogs look very similar to certain nude images that people take of themselves and post/share online. So having a classifier that could detect this with high probability and warn the user that the image could be inappropriate could be useful. Instant messengers, Snapchat, or online forums like Reddit where users can post and send pictures could all use this. The model should be change more so it is biased to have more false-negatives than false-positives. This way it's more likely to catch not phallic objects than to accidentally let them past and sent to a user without warning.

VIII. Conclusions

This project should allow a classifier that can predict hotdogs with good accuracy. I think with all the new images I would create and with going back and alerting the dropout rate, number of hidden layers in the network, and the changing the image set I could get my model as accurate as possible under the limitations of the small starting image set. CNNs do better with large amounts of training data so even with the image transformations I think that would be the limiting factor for this program. Despite this I think this is a great starter project and although I was not able to do it for real for this assignment, I think I learned enough and made a clear enough roadmap that I could do it. So, I am playing on doing it after I finish this semester. This has given me a much better understanding of TensorFlow and Keras than I previously had, and I am excited to use these libraries more in the future.

References

[1] Nilesh Barla *Self-Driving Cars with Convolution Neural Networks (CNN)*, Wednesday December 1st, https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn

[2] Rajan Patel, *Giving Lens New Reading Capabilities in Google Go, Google AI Blog*, Wednesday December 1st, 20201 https://ai.googleblog.com/2019/09/giving-lens-new-reading-capabilities-in.html

[3] Dans Becker, *Hot Dog – Not Hot Dog*, November 2021, https://www.kaggle.com/dansbecker/hot-dog-not-hot-dog

[4] John Terra, *Keras vs TensorFlow vs Pytorch: Key differences among the deep learning framework*, Oct 28, 2021. https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

[5] Claire D. Costa, March 24, 2020, https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

[6] Jason Brownlee, July 24, 2017, *How Much Training Data is Required for Machine Learning*, Machine Learning Masterty, https://machinelearningmastery.com/much-training-data-required-machine-learning/

[7] *Just an example, but the *TensorFlow with GPU* example on Google Colab https://colab.research.google.com/notebooks/gpu.ipynb

[8] The Pillow Library https://pillow.readthedocs.io/en/stable/

[9] https://www.tensorflow.org/guide/data#batching_dataset_elements

[10] https://keras.io/about/

[11] Loannis Nasios, *Keras: concatenate two images as input*, March 2nd 2018, https://stackoverflow.com/a/49065224

[12] https://www.tensorflow.org/api_docs/python/tf/data/Dataset#cache, Accessed Monday, December 13th, 2021

[13] mrry, *How to cache data during the first Epoch correctly, https://stackoverflow.com/a/50536314*

[14] Aniruddha Bhandari, *Image Augmentaation on the fly using Keras ImageDataGenerator*, August 11th 2020, https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/#

[15] Aniruddha Bhandari, *Image Augmentaation on the fly using Keras ImageDataGenerator*, August 11th 2020, https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/#

[16] *How to Normalize, Center, and Standardize Image Pixels in Keras*, 25 February 2021, https://www.geeksforgeeks.org/how-to-normalize-center-and-standardize-image-pixels-in-keras/

[17] tf.keras.preprocessing.image.ImageDataGenerator, Tuesday December 14th. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[18] The table of activation functions (This is just an example as Wikipedia is not a credible source. https://en.wikipedia.org/wiki/Activation_function#Folding_activation_functions

[19] Shuaib Ahmed, *How to know whether the data is linearly seperable*, June 11th 2020, https://stats.stackexchange.com/a/182378

[20] Shuaib Ahmed, *How to know whether the data is linearly seperable*, June 11th 2020, https://stats.stackexchange.com/a/182378

[21] *The Sequential Model*, Tuesday Dec 14th 2021, https://www.tensorflow.org/guide/keras/sequential_model

[22] *The Sequential Model*, Tuesday Dec 14th 2021, https://www.tensorflow.org/guide/keras/sequential_model

[23] *GeeksforGeeksChoose optimal number of Epochs to train a neural network in Keras*, 8 June, 2020. https://www.geeksforgeeks.org/choose-optimal-number-of-Epochs-to-train-a-neural-network-in-keras/

[24] *GeeksforGeeksChoose optimal number of Epochs to train a neural network in Keras*, 8 June, 2020. https://www.geeksforgeeks.org/choose-optimal-number-of-Epochs-to-train-a-neural-network-in-keras/

[25] *Sklearn.score*, Tuesday December 14th, 2021, https://scikit-learn.org/stable/modules/model_evaluation.html

[26] *Dropout Layer*, Keras, Tuesday December 14 2021, https://keras.io/api/layers/regularization_layers/dropout/

[27] Saikat Roy, *Does adding more layers always result in a more accurate convolutional neural network*, https://www.quora.com/Does-adding-more-layers-always-result-in-more-accuracy-in-convolutional-neural-networks