Hot-dog or Not Hot-Dog: When Life Imitates Art

Presented to

CSUF

COLLEGE OF

Natural Sciences and Mathematics

California State University, Fullerton

Math 533 Fall 2023

Dr. Sam BEHESTA

Prepared by

Emilio VASQUEZ

December 17, 2023

Contents

10 Appendix

11 Python Code

1 Abstract

As the days progress, artificial intelligence (AI) encapsulates more and more of our daily lives stepping closer to the researcher's dream of achieving artificial general intelligence (AGI), or human like level of intelligence. Key figures in the AI race such as OpenAI CEO Sam Altman says that "GPT-4 or GPT-5 would have been considered AGI to a lot of people ten years ago" and people these days may say GPT-5 is a "nice little chatbot or whatever" ("OpenAI CEO Sam Altman and CTO Mira Murati on the Future of AI and ChatGPT | WSJ Tech Live 2023."). The gap in the way computers interpret the world in a similar way as we humans do grow closer. This analysis explores one sub domain of AI known as computer vision which involves training a computer to see the world as we see it although it can be argued in the opposite direction as well. The central premise of this report is to imitate a gag from the show "Silicon Valley" to train models to classify images into either being a 'Hot-Dog' or 'Not Hot-Dog' using 3,000 images. The training dataset was split evenly between the two classes.

To address this topic, logistic regression and Convolutional Neural Networks (CNN) were employed. Three different logistic models were created in which they classified the binary class as a function of some transformation of the the average red-green-blue (RGB) value of the pixels found within each image. Utilizing the paper on AlexNet, the CNN structure of AlexNet was implemented using tensorflow. A pre-trained CNN for VG19 was utilized as well through tensorflow. Of the five models that were created, the pretrained VG19 model performed the best in achieving an accuracy of around 92% when using accuracy of test set prediction as the objective measure of fit.

2 Introduction

AI has come a long way since the start of the millennium and is in all of our lives whether we are fully aware of it or not. From browsing content on a video streaming service, taking photos on our smartphones, and to the advancements in the automotive industry with autonomous vehicles, AI plays and will continue to play a pivotal role in shaping the rest of our lifetimes and centuries to follow.

Recommendations provided by our favorite video streaming services that are based on our viewing habits exemplify the unavoidable nature of AI. Sophisticated algorithms analyze all of our preferences and viewing habits to curate a personalized selection of movies, shows, and video content with the end-goal to keep the user scrolling, engaged and within a company's digital sandbox. This exemplifies the impact of AI on user engagement and satisfaction in the entertainment realm.

Everyone has captured photos on their smartphone before. The striking thing is that users of such devices have already used the fruits of AI without even knowing! Images taken on phones make use of AI as written by Berrada in her article on her Google blog to deliver photos that are sharp, crisp, and color tones that are pleasing to the human eye. Most editing done after the fact employ AI to ensure realistic and satisfying results to the end user (Google).

On an even grander scale, AI extends into the realm of autonomous vehicles, marking a shift in the automotive landscape. The growing adoption of electric vehicles throughout the United States increased over the years (Randall). This adoption of electric vehicles will only continue to propel autonomous driving to the point where there may not ever be a need for human intervention to get from point A to point B. At the center of this big idea with taking images on a phone and autonomous vehicles is a subsection of AI known as computer vision.

Computer vision is defined as "A field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs" (IBM). Within the realm of computer vision, this report concentrates on a specialized subdomain known as image classification. Image classification involves training a computer on a set of images that are all each associated with unique class labels. Through this training process, a computer will develop the ability to categorize new images that are fed into it into the trained model.

2.1 Goals

The goal of this analysis and report is to re-create HBO's "Silicone Valley" hot-dog, not hot-dog' gag in season 4 episode 4. In the episode prior, the show's fictional company Pied Piper saw it's venture capitalist Erlich Bachman in a meeting room with one of the show's side characters Jian Yang pitching an app known as 'Seefood' which was presented to operate like the music identifying app Shazam, but for food. Users were supposed to be able to provide photos of their food and it would classify what that food is and provide nutritional facts. The next episode is the culmination of Jian Yang's hard work at building the Seefood app to which everyone's shock can only classify food into either a 'hot-dog' class or 'not hot-dog' class.

What started off as a television show gag is now a graduate student's machine learning topic, this report will delve into methodologies to building a binary 'Hot-Dog' and 'Not Hot-Dog' classifier through the following modeling techniques:

- Logistic Regression
  - Rescaling average RBG values to between 0 and 1 instead of 0 to 255
  - Standardize the average RGB values with standard normal
  - No rescaling of the average RGB values
- Convolutional Neural Network
  - AlexNet
  - VG19

3 Structure of Data

3.1 Data Description

This analysis uses a published image dataset from Yashvardhan Jain on Kaggle, a well known website that hosts data for data science projects. This dataset was built specifically for the task at hand. Luckily, there were many other people who attempted this project and few articles on this such as the one that I stumbled upon from Towards Data Science. It contains 3,646 images between two classes: hotdog and not hot-dog. In addition, it contains a pre-constructed train and test set. Within the train set, there are 1500 images that belong to the hot-dog set and 1500 that pertain to the not-hot-dog set. The test set contains 323 hot-dog images and 323 not-hot-dog images.

3.1.1 Pixel Count of an Image

A digital image in and of itself is a matrix collection of pixels (Baldwin). If one has ever shopped for device with a screen of some sort, chances are that they have come across terms such as "HD", "4K", and sometimes maybe even "8K". All these are references to how many pixels are in the screen. Many new TVs sold on the market today are solely 4K just by strolling around an electronic store, which means the screens can display images 2,160 pixels tall and 3,840 pixels wide.

Examining a single photo from the dataset, one can see that a single photo from this dataset is of the size 299 by 299.

Figure 1. Dimensions of a single image from the train dataset

As one can see, these photos are drastically smaller than what one would consider when buying a TV. In fact, the old tube CRT TVs had a greater resolution than this at 480p (Trevor the TV), or 640x480. The images are small with good reason: to improve computation time. In the algorithms and methods that proceed, working with a large image increases computation time drastically.

Taking this a step further into the beginnings of exploratory data analysis, a total of 3,646 images were read into a pandas dataframe to verify the resolution of the entire dataset.

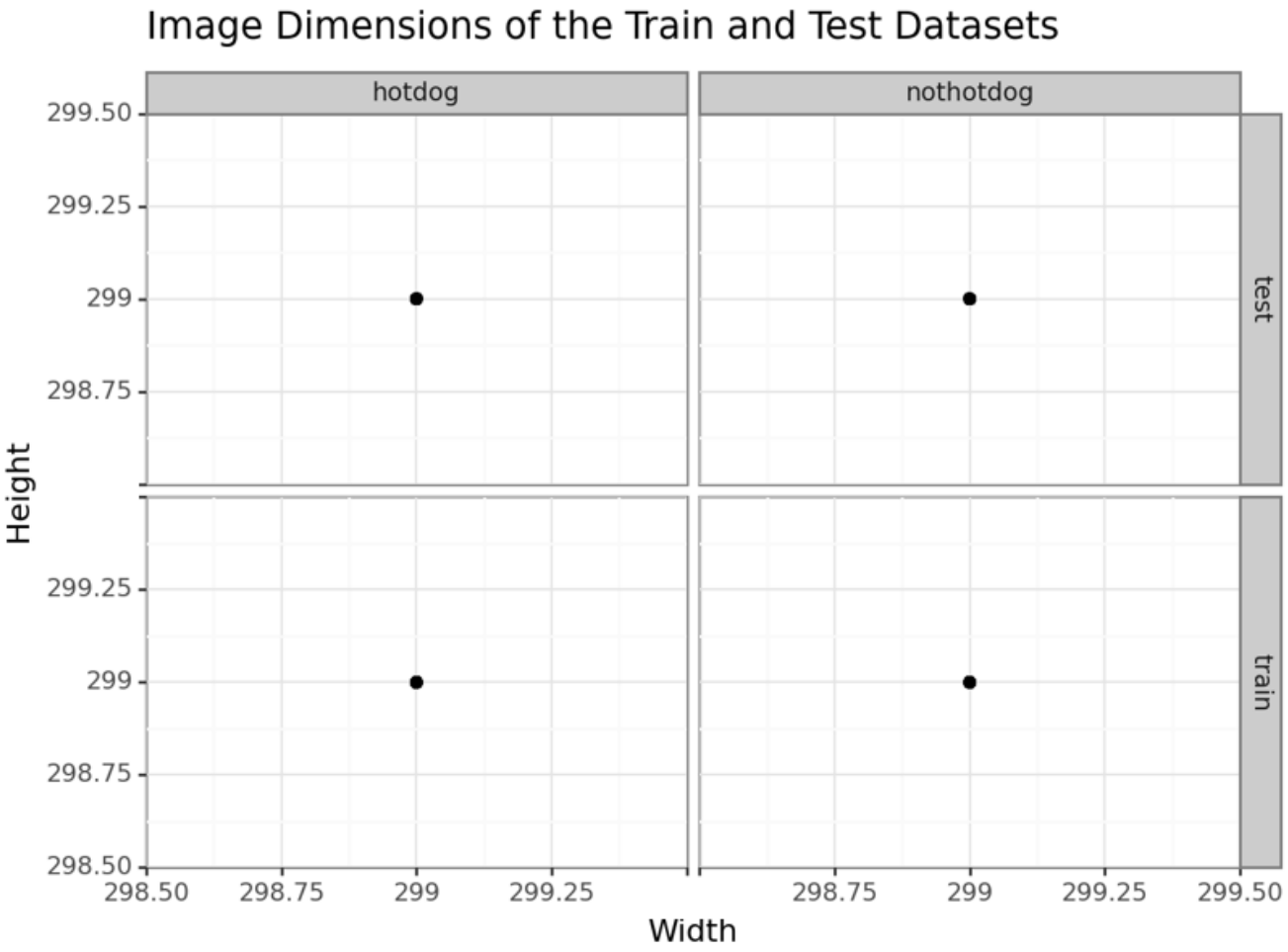

Figure 2. Scatterplot of image sizes for all images in the dataset

In examining the figure above shows a scatterplot of the size of all images within the dataset. In both the train and test set, it appears that all 3,646 images are uniform as they are all centered at 299x299. Had the images not been uniform in resolution, there would have been a lot more points on the plot.

3.1.2 RGB of a Pixel

The above explored the resolution of all images in the dataset but for the purposes of this report, we can go one level deeper on the structure of the image through understanding the red-green-blue nature of image. As mentioned before, each image is made up of pixels. Within each pixel contains the color of that pixel which is a blend of a red, blue, and green channel (Baldwin). Each channel begin spans from 0 to 255 with values closer to 255 being more intense. Being that there exists three channels, any image can be broken down into a red, green, and blue channel as shown below.

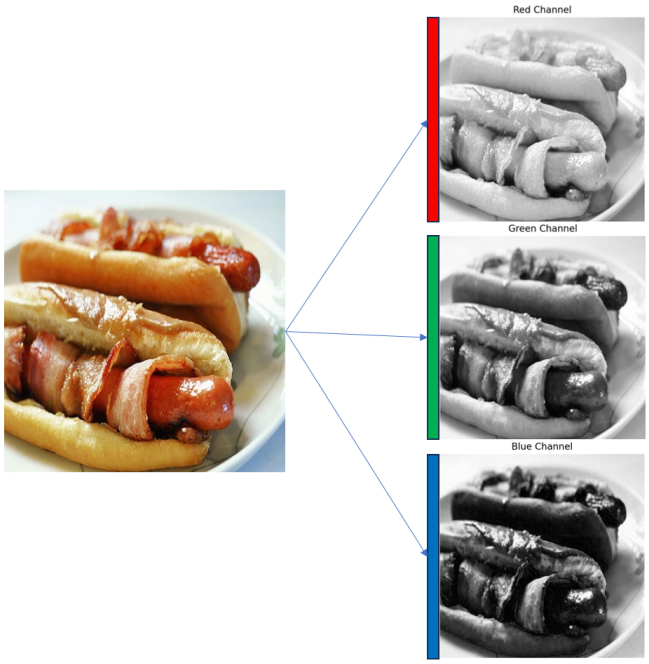

Figure 3. Breaking down a hot-dog image into the three channels

In splitting the image above, one can see how much each channel contributes to the overall image output. Visually one can gain information on that intensity of a specified color in each pixel of the image. The lighter the image, the higher the contribution that layer has to that color. In this specific example, one can see that the red layer is the lighest of the three which makes sense because the hot-dog weiner themselves are inherently red. This theory lays the foundation for the exploratory data anlaysis that follows.

4 Exploratory Data Analysis

Previous sections delved into the structure of image as applied to the dataset at hand, but now we direct our attention to a deeper analysis on the actual train and test set of the two classes. The goal of this analysis is to create an image classifier that classifies images into either the hotdog class or the not hot-dog class.

4.1 Exploring the "Hot-Dog" Class

Figure 4. A random selection of images from the hot-dog class

In the figure above, one can see a snapshot of the photos that are within the hot-dog class. There is a wide assortment of photos of hot-dogs in this small random sample. There are hot-dogs with simply ketchup and mustard to more complex hot-dogs such as the one that is in the bottom right quadrant. To a trained food connoisseur myself, that hardly looks like a hotdog to myself but the author of that dataset says this is a hotdog. Interesting to see that in some images such as the one in the center, there is more to an image than just a hotdog as there are chill cheese fries.

Also interesting to see that there exists almost cartoon and clip-art like images of hotdogs as well as seen in the photo that in the bottom center. Randomly cycling through these photos, there were two other photos of similar nature. Lastly, this same picture also highlights that in the dataset there exists photos of varying resolution. This can be seen as this particular photo is much more blurry when compared to the rest, which are signs that this image was a lot smaller than the 299x299 resolution an was made larger in the curation of this dataset.

Delving a bit deeper into the actual RGB values of the hot-dog class, the below figure presents a distribution of pixels that fall within the three respective RGB channels.

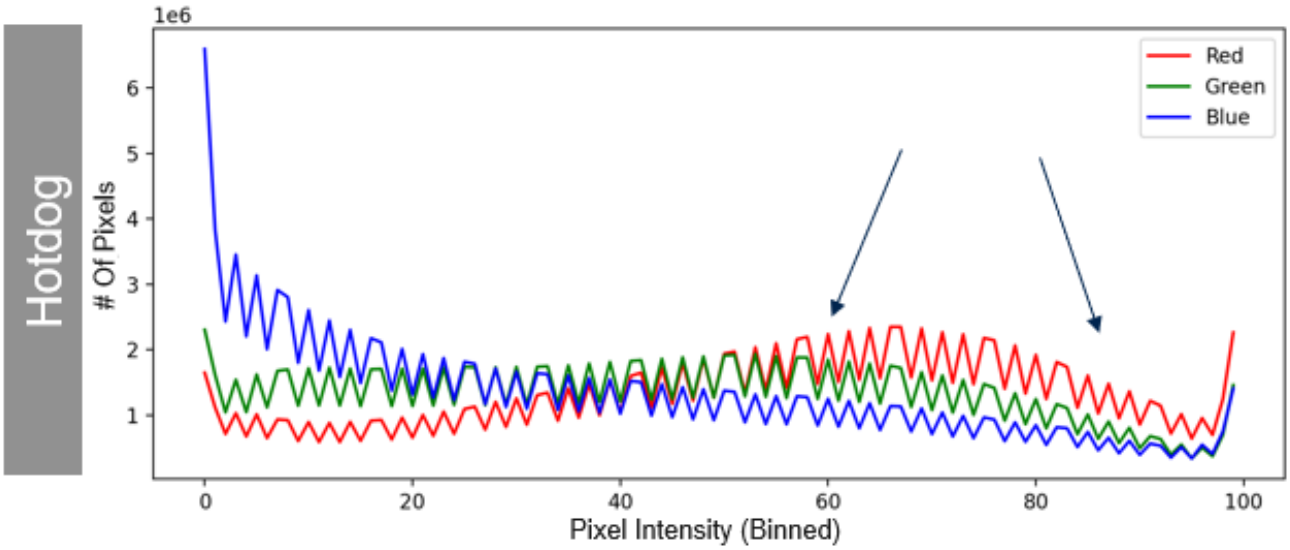

Figure 5. RGB pixel intensity for the hot-dog class

As the pixel intensity increases, one can see that the red channel overshadows the green and blue channel. Specifically, we are looking between the 60 to 80 bins. This may be a factor of the red wiener found in every hot-dog.

4.2 Exploring the "Not Hot-Dog" Class

Figure 6. A random selection of images from the not hot-dog class

In the figure above, one can see a snapshot of the photos that are within the not hot-dog class. There is a mix of photos of other foods, inanimate objects, and people. In the top center, one can see a group of friends hanging out. Middle row to the right shows a feret. The rest of the photos are either of furniture or other types of food that are not hot-dog.

Delving a bit deeper into the actual RGB values of the not hot-dog class, figure 7 below presents a distribution of pixels that fall within the three respective RGB channels. Comparing figure 7 for the not hot-dog class below to the figure 5 above for the hot-dog class, there is not much of a peak for the red channel for the pixel intensity in the between 60 and 80. Looking at figure 7 specifically, there is actually not much of a peak at all for the red channel. Blue levels are more or less the same as compared to the hotdog class. Green channel intensity is slightly lower overall.

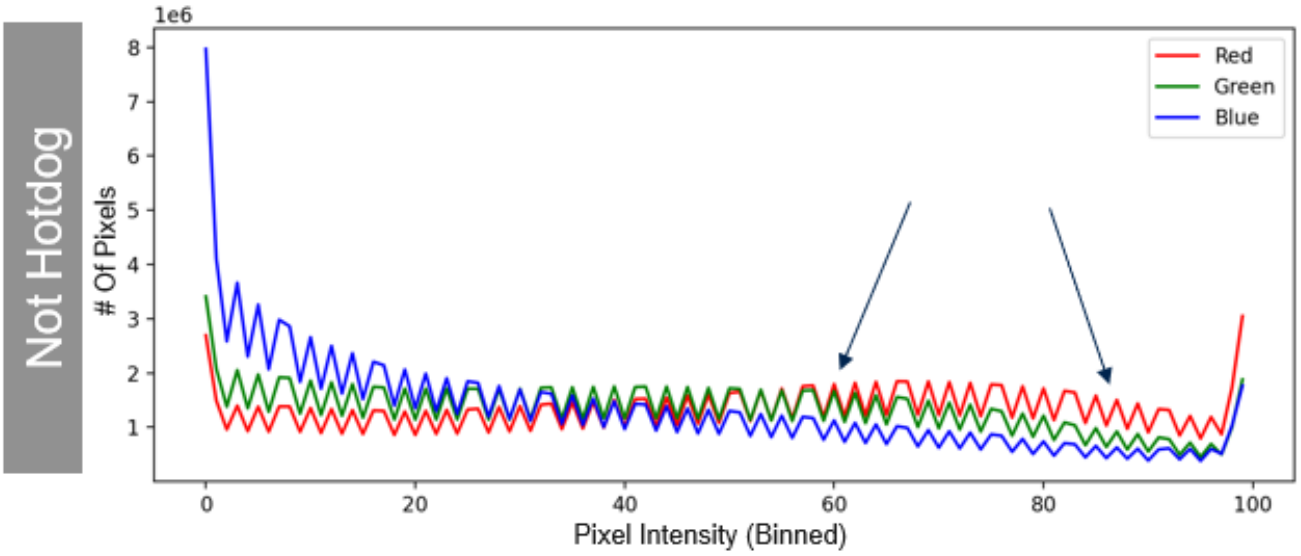

Figure 7. RGB pixel intensity for the not hot-dog class

4.3 Data Preparation

4.3.1 Resize, Augment, and Rescale

The first step to most image classification tasks is to ensure that the photos are all of the same size. The author of the provided Kaggle dataset already performed this step and the dimensions of all the photos were verified in earlier sections.

Next would be to perform image augmentation. Convolutional neural networks models allow for artificial augmentation to increase the training sample size which in theory will results in better ability to train a model (Gareth, et al.). Figure 8 below shows the many ways all the images in the train set were transformed before being fed into the model. Each image underwent the process of flipping, rotating, and zooming in and out. This augmentation acts as a forum of regularization. The convolution neural network is exposed to variations of the same image which force the model to learn more robust features and prevents it from simply memorizing specific patterns present in the training set.

Lastly, the final step in pre-processing is the act of image rescaling. As mentioned previously, pixels range from a value between 0 to 255. In dividing the entire dataset by 255, the new range of a pixel intensity value goes now between 0 and 1. This in combination with the resizing above allow for a drastically faster computation within the CNN. This resizing specifically allows for greater compatibility with different activation functions such as the sigmoid function as well.

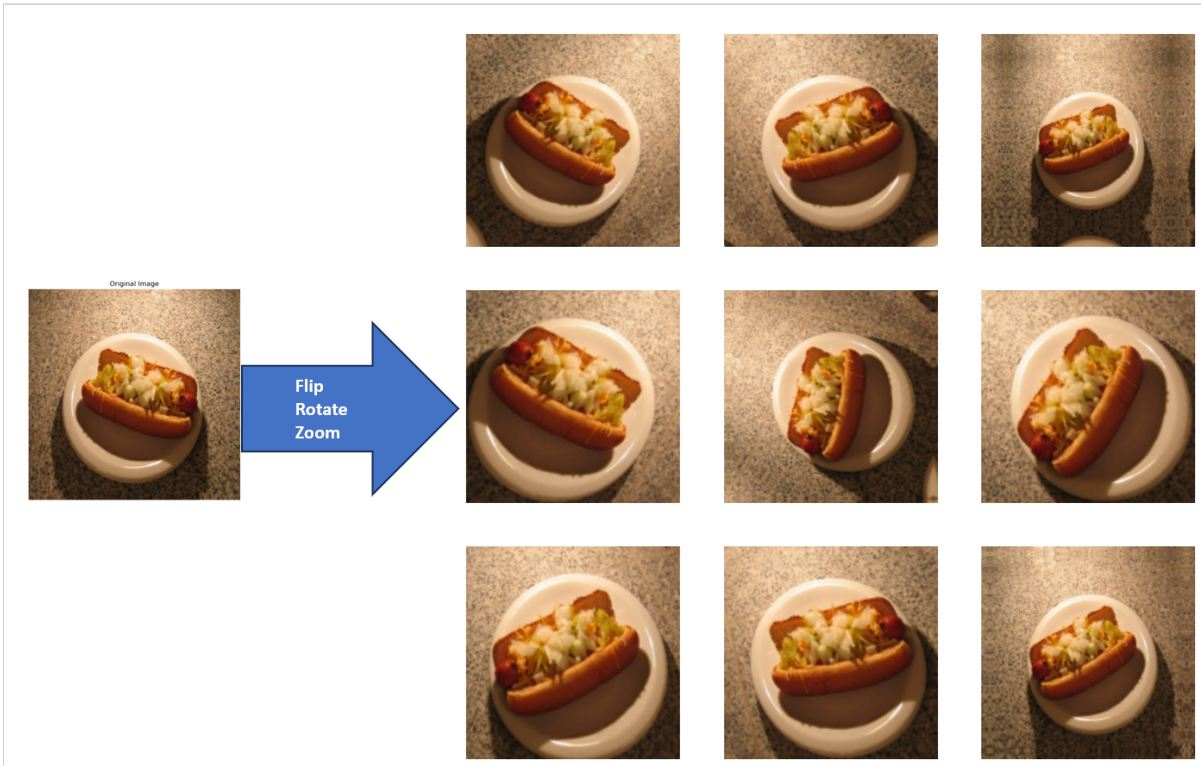

Figure 8. Augmenting the hot-dog training data

5 Methods

In this research to address the goals stated earlier, we have considered two types of modeling approaches.

- Logistic Regression
- Convolutional Neural Networks

To determine which model performed the best, the accuracy of the model on a single test set was used.

5.1 Logistic Regression

Since the classification is currently for a binary class, upon the recommendation of Dr. Sam Behesta from California State University, Fullerton, a logistic regression model was implemented. Utilizing the extensive work completed above in reading all the images into a dataframe and exploring the RGB values, a logistic regression model was created around using solely the average RGB values for every photo.

There were three variations explored but they all fell shy of any usable performance. The final three models created were

- **Model 1**: Utilized RGB predictors that were rescaled between 0 and 1

$$\text{Model 1: } \widehat{class} = -2.7982(\text{avg red}) + 2.2402(\text{avg green}) - 0.8915 * (\text{avg blue})$$

- **Model 2**: Utilized RGB predictors that were rescaled to have a mean of 0 and a standard deviation of 1

$$\text{Model 2: } \widehat{class} = -0.6626 * (\text{avg red}) + 0.7006 * (\text{avg green}) + -0.3331 * (\text{avg blue})$$

- **Model 3**: Utilized RGB predictors as they were. No rescaling was performed.

$$\text{Model 3: } \widehat{class} = -0.0197(\text{avg red}) + 0.0238 * (\text{avg green}) - 0.0106 * (\text{avg blue})$$

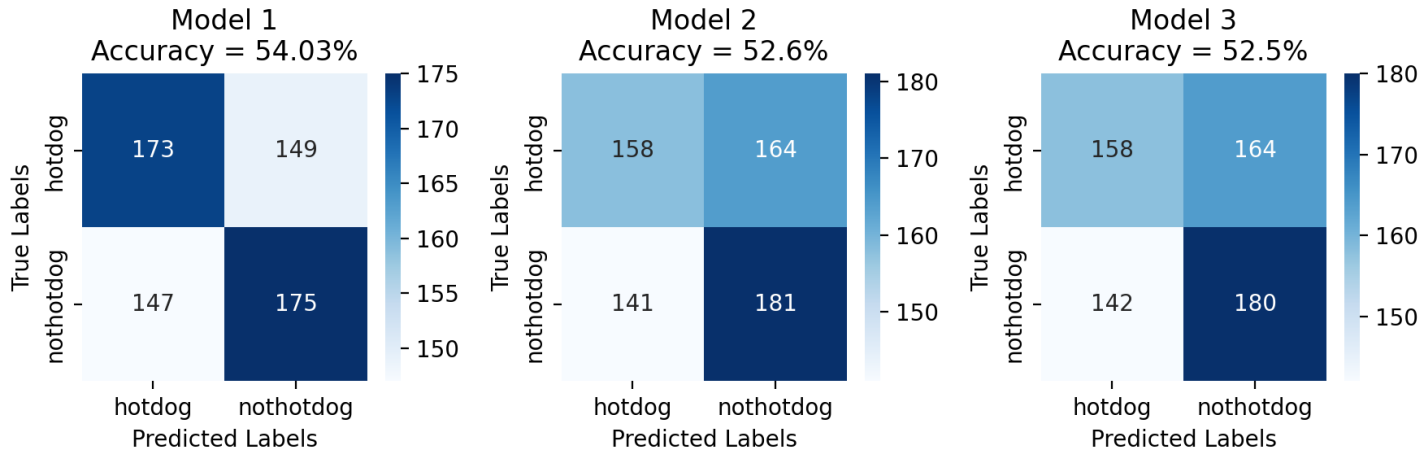

Figure 9. Comparing the three different Logistic Regression Models

In comparing the three models, model 1 performed the best of the three logistic regression models where the RGB values were scaled between 0 and 1 performed the best. Overall however, the difference is almost negligible given how close all the final accuracy results are. In the end, the results were not much better than a coinflip at deciding whether or not an image was a hot-dog or not.

5.2 Convolutional Neural Networks

The next couple of models used within this analysis are convolution neural networks which are the best class of models for computer vision tasks due to the way they process images. In referencing An Introduction to Statistical Learning (ISLR), CNNs are commonly used in image recognition by identifying smaller features of an overarching image. By identifying these smaller features, a final classification can be made to determine the object in the image, similar as shown in the image below from the text.

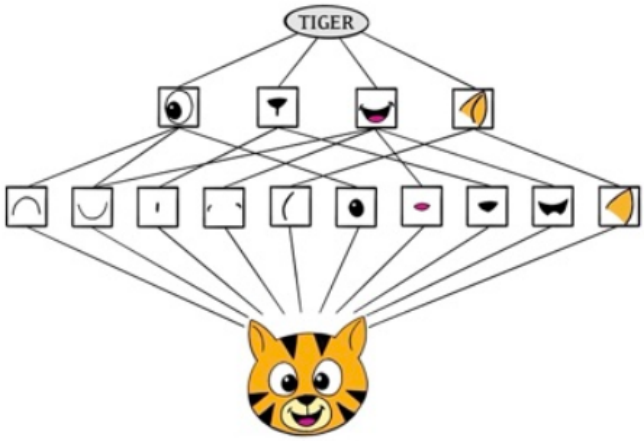

Figure 10. Diagram on how a CNN classifies an image from the ISLR textbook

Within this network there are convolution layers and pooling layers. Within a convolution layer, many convolution filters are applied to the image with each producing a feature map where ReLU activation is typically applied. A pooling layer such as max pooling summarizes non-overlapping blocks (such as 2 x 2) by their max value. In other words, it's a way to reduce dimensions while keeping the most prominent features. The last layer can have a softmax activation when needing to predict multiple classes.

This analysis specifically will review the results of two particular CNN:

- AlexNet
- Pretrained VG19

5.2.1 AlexNet

The first CNN model created was the closest implementation of AlexNet that could be done in tensoflow. AlexNet was first introduced to the world in 2012 by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. It was one of the first CNN's to utilize "dropout" regularization. It is through this act of randomly dropping or ignoring selected nodes in the CNN with a certain

probability where the ever-lasting issue of overfitting is curtailed to some extent. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge in 2012 where it was able to achieve an error rate of 15.3% which is a huge improvement over the 26.2% achieved by the second best entry. Below is the structure of Alexnet.

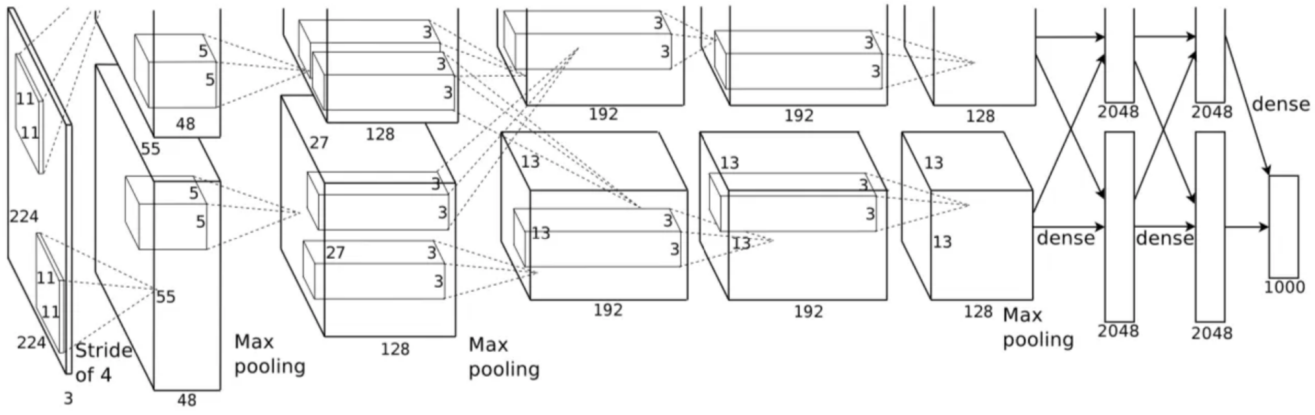

Figure 11. Structure of AlexNet CNN from the published paper

In brief, AlexNet contains 5 convolutional layers with 3 fully connected layers at the end. Max pooling is applied after the first, second, and fifth convolutional layers. The implementation of thiss requires the training of 62,379,345 parameters. It is not a deep neural network by today's standards, but saw great success over a decade ago which led me to try out this model first.

5.2.2 VG19 Pretrained Via Transfer Learning

VG19 is another widely used CNN that gets it's name from it being 19 layer deep which is already double the size of AlexNet. 16 of these layers are convolution layers with 3 fully connected layers. Visually, one can view the general structure of a general VG19 model below in figure 12.

Unlike the previous implementation of AlexNet, this implementation of VG19 uses a pre-trained model to explore how well a pre-trained model might perform. This particular model was trained on millions of images from ImageNet which is vastly more from the model trained above. Using a pre-trained model allows one to re-use weights and configurations that were determined using large datasets by people with better computational power. It is a quicker way to allow individuals to get better performance and results as a base-line.

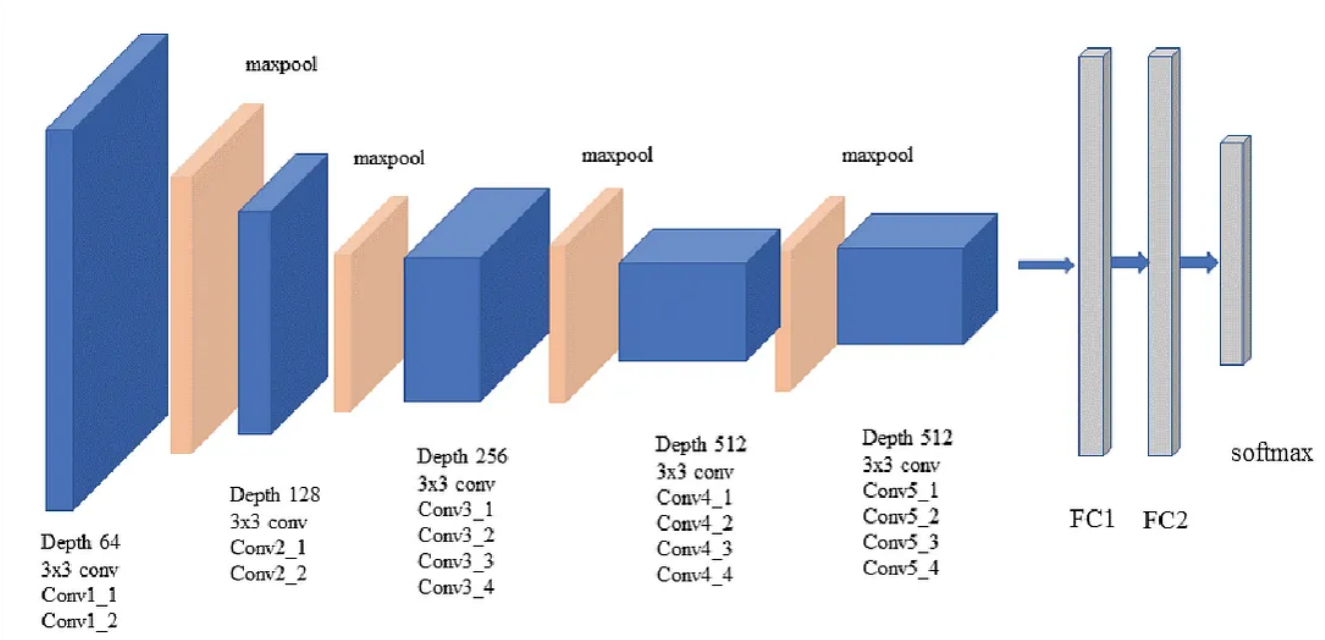

Figure 12. Structure of a general VG19 CNN

5.2.3 Results of CNN

Between the two convolutional neural networks, VG19 performed significantly better than AlexNet

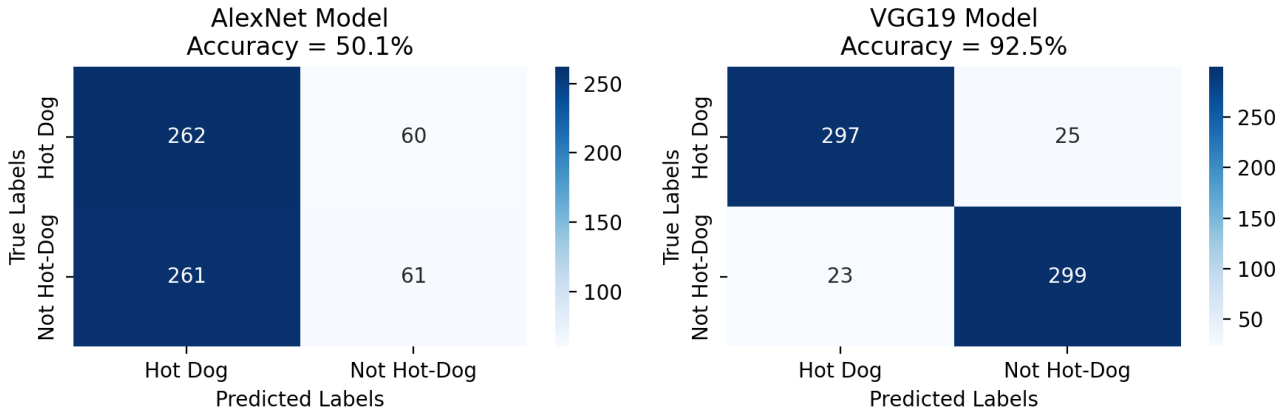

Figure 13. CNN model accuracy results

AlexNet had an accuracy of 50.1% where as VG19 had an acurracy of 92.5%. AlexNet particularly failed at classifying the Not Hot-Dog class, only classifying correctly 61 of the 644 total test images. The VG19 was able to clasifify Not Hot-Dog a lot more frequently at 299 correct.

6 Results

Below are the results of the 5 models combined into one visualization.

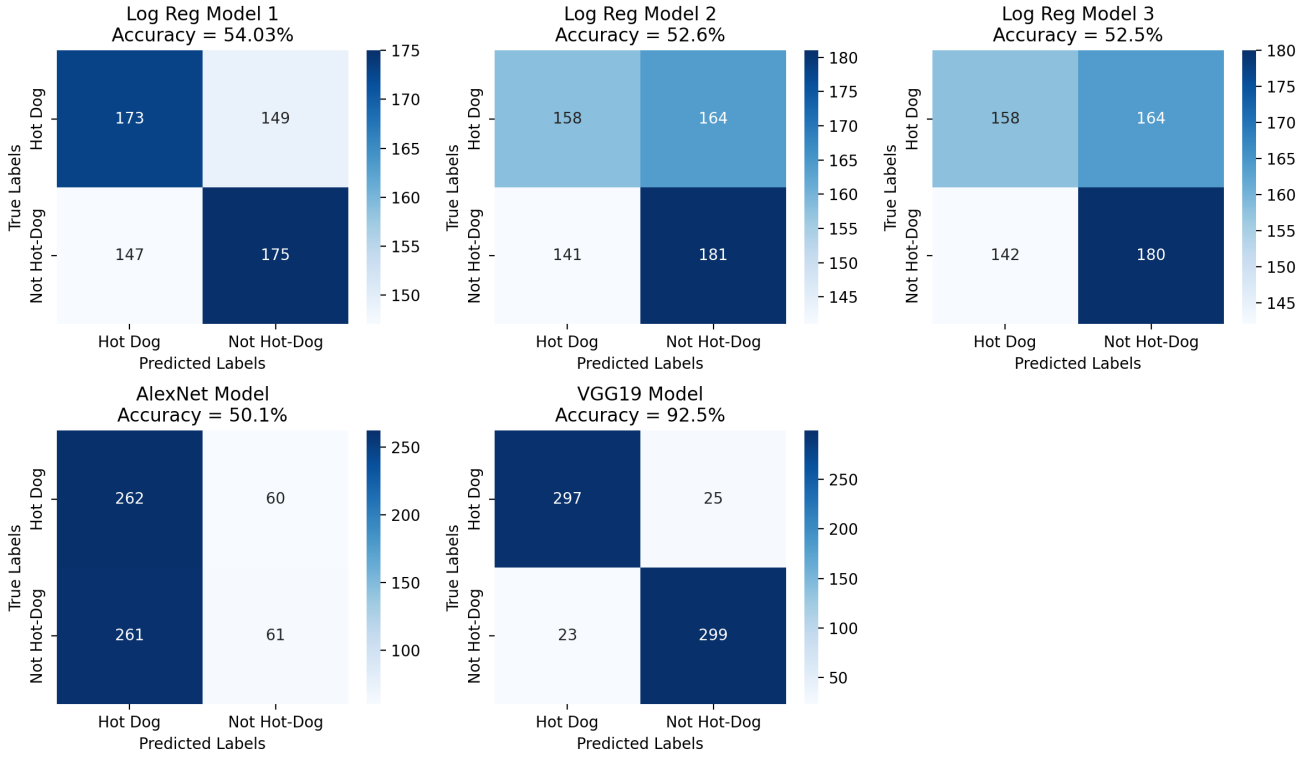

Figure 14. All five model results visualized

The analysis above shows that CNN is the route to go. Adding more layers through the VG19 implementation increased accuracy on the test data greatly over the AlexNet implementation. However, it is difficult to discern whether or not this is actually because of more layers or the fact that it was pre-trained via million of images from ImageNet. Furthermore, since the dataset was trained on images from ImageNet, it is possible there is some overlap in the data used to train the model and the test dataset.

All three Logistic Regression surprisingly did well compared to the AlexNet implementation, which was a largely unexpected result. It was able to beat out a much more complex model such as a CNN by simply using the average RGB values for each image as a predictor. This is extremely simplistic compared to how a CNN operates which can take into account various shape(s) of items found within an image.

7 Summary and Future Work

Convolutional neural networks proved to be the be recommended machine learning technology to apply to image classification. This is largely due to the ability to apply many convolutional layers which allowed to capture hierarchical features in the data. CNNs are designed for tasking involving grid-like data, which is all what an image is as demonstrated in sections above. Within these images are complex relationships of shapes, color, all defined by collections of individual pixels. It was shown that simply looking at the RGB values of images that there is a story to be told within those values at least when it came to classifying hot-dog or not hot-dog.

For future work, future data scientists would benefit from exploring other neural networks with this dataset. Some neural networks that were discovered at the time of this writing were the ReseNet50 and GoogLeNet/Inception models which utilize 50 and 48 layers respectively. Due to computing constraints, these models were not able to be run within the provided timeline.

Secondly, this project could be further extended in the data wrangling aspect. Recall, this analysis was completed using a dataset from Kaggle that had a total of 3,646 images to train and test on. For any sort of image classification task, this can be considered small. To extend this project, one could build a web-scrapper using python packages such as selenium to scrape the web for even more hot-dog pictures and not hot-dog pictures. From there, one could use this larger dataset to train a VG19 model from the ground up in a similar fashion the way the AlexNet model was trained, computational power permitting. This would be the true test to see if the VG19 transfer learning model performed so well because of the additional layers or because it was pre-trained on millions of images. Additionally, using this hypothetical larger dataset through the act of web scraping, it would be great to be able to implement additional classes of foods to better fulfill Erlich Bachman's initial pitch of the Seefood app.

Lastly, it would be great to further delve into the possibilities of why the logsitic regression model was able to perform as well as it did. Investigations under the deadline of this report were able to verify the results to the best understanding of the author but it is possible that there exists some code error that led to this unexpected result. It may have occurred in the processing of rgb values into a pandas dataframe. It is possible as well that the implementation of the AlexNet model was not correct or some weird bug that could be explored by a future individual.

8 References

Baldwin, Doug. "Introduction to Digital Images." Digital Images, 30 Aug. 2023, www.geneseo.edu/~baldwin/reference/images.html.

Berrada, Dina. "4 New Google Photos Features on Pixel 8 and Pixel 8 Pro." Google, Google, 4 Oct. 2023, blog.google/products/photos/google-photos-features-pixel-8-pro/.

Chouhan, Sanjay. "hot-dog or Not hot-dog." Medium, Towards Data Science, 26 Feb. 2022, towardsdatascience.com/hot-dog-or-not-hot-dog-ab9d67f20674.

DanB. "Hot Dog - Not Hot Dog." Kaggle, 19 July 2018,www.kaggle.com/datasets/dansbecker/hot-dog-not-hot-dog.

Hewage, Roland. "Extract Features, Visualize Filters and Feature Maps in VGG16 and VGG19 CNN Models." Medium, Towards Data Science, 15 May 2020, towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0.

Jain, Yashvardhan. "Hotdog-Not-Hotdog." Kaggle, 17 June 2020,

kaggle.com/datasets/yashvrdnjain/hotdognothotdog.

James, Gareth, et al. *An Introduction to Statistical Learning: With Applications in R.* Springer, 2021.

Krizhevsky, Alex, et al. "ImageNet classification with deep convolutional Neural Networks." *Communications of the ACM,* vol. 60, no. 6, 2017, pp. 84–90, https://doi.org/10.1145/3065386.

M. Masum, PhD. "Applying the VGG Pre-Trained Model with Keras: A Poor Example of Transfer Learning." Medium, Towards Data Science, 16 Apr. 2022, towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a.

"OpenAI CEO Sam Altman and CTO Mira Murati on the Future of AI and ChatGPT | WSJ Tech Live 2023." YouTube, Wall Street Journal, 21 Oct. 2023, https://www.youtube.com/watch?v=byYlC2cagLw. Accessed 3 Dec. 2023.

Pleticha, Meghan. "Teambuilding Exercise." Silicon Valley, season 4, episode 4, HBO, 14 May 2017, Accessed 16 Oct. 2023.

Randall, Tom. "California Shows an Electric-Car Uprising Headed for the US." Bloomberg.Com, Bloomberg, 8 Sept. 2023, www.bloomberg.com/news/articles/2023-09-08/california-shows-an-

electric-car-uprising-headed-for-the-us.

Rosebrock, Adrian. "Clever Girl: A Guide to Utilizing Color Histograms for Computer Vision and Image Search Engines." *PyImageSearch,* 17 Apr. 2021, pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines/.

Simonyan, Karen, and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition,* 4 Sept. 2014, pp. 1–14, https://arxiv.org/pdf/1409.1556.pdf.

Trevor the TV. "History of TV Pixels: From CRT to 8K: Electronic World Blog." Electronic World TV Blog, 3 Feb. 2020, www.electronicworldtv.co.uk/blog/history-of-pixels#:~:text=Pixels%20in%20CRT%20screens%20worked,onto%20the%20screen%20in%20lines.

Wei, Jerry. "Alexnet: The Architecture That Challenged CNNS." Medium, Towards Data Science, 25 Sept. 2020, towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

"What Is Computer Vision?" IBM, www.ibm.com/topics/computer-vision. Accessed 16 Nov. 2023.

Wunk. "Average Color of an Image, Vectorized." DEV Community, DEV Community, 5 May 2019, dev.to/wunk/average-color-of-an-image-vectorized-4802.

9 Acknowlwedgement

This was an insightful project to an already extremely insightful course. Thank you very much being a great professor who takes great pride in teaching the subject matter. I think I was able to get my new role as a Forecasting Analyst at Southern California Edison partially due to the homework/projects done in this class that I posted to GitHub, so thank you very much for a wonderful semester. I've learned a lot and now have a set of textbooks I can refer to whenever I might want to delve further into any given machine learning topic.

10 Appendix

10.1 Verifying class size

Figure 1. Verifying the class distribution

After reading in all the images, the above figure verified the author's readme in the size of the train and test sets for each class.

10.2 AlexNet Diagnostics and Readouts

10.2.1 Model Summary

```
Model: "AlexNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image (InputLayer)          [(None, 227, 227, 3)]     0

 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       614656

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 256)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 384)       885120

 conv2d_3 (Conv2D)           (None, 13, 13, 384)       1327488

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       884992

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 1000)              4097000

 dropout_2 (Dropout)         (None, 1000)              0

 dense_3 (Dense)             (None, 1)                 1001

=================================================================
Total params: 62379345 (237.96 MB)
Trainable params: 62379345 (237.96 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2. Model summary for the AlexNet model

10.2.2 Model History

```
Epoch 1/10
75/75 [==============================] - 95s 1s/step - loss: 2.3834 - accuracy: 0.4921 - val_loss: 0.6870 - val_accuracy: 0.515
0
Epoch 2/10
75/75 [==============================] - 98s 1s/step - loss: 0.6912 - accuracy: 0.5304 - val_loss: 0.6839 - val_accuracy: 0.573
3
Epoch 3/10
75/75 [==============================] - 95s 1s/step - loss: 0.6853 - accuracy: 0.5575 - val_loss: 0.6988 - val_accuracy: 0.538
3
Epoch 4/10
75/75 [==============================] - 96s 1s/step - loss: 0.6819 - accuracy: 0.5663 - val_loss: 0.6819 - val_accuracy: 0.561
7
Epoch 5/10
75/75 [==============================] - 100s 1s/step - loss: 0.6562 - accuracy: 0.6200 - val_loss: 0.6386 - val_accuracy: 0.62
67
Epoch 6/10
75/75 [==============================] - 91s 1s/step - loss: 0.6376 - accuracy: 0.6492 - val_loss: 0.6508 - val_accuracy: 0.618
3
Epoch 7/10
75/75 [==============================] - 92s 1s/step - loss: 0.6107 - accuracy: 0.6662 - val_loss: 0.6561 - val_accuracy: 0.600
0
```

Figure 3. Training history for AlexNet

10.2.3 Training and Validation Accuracy

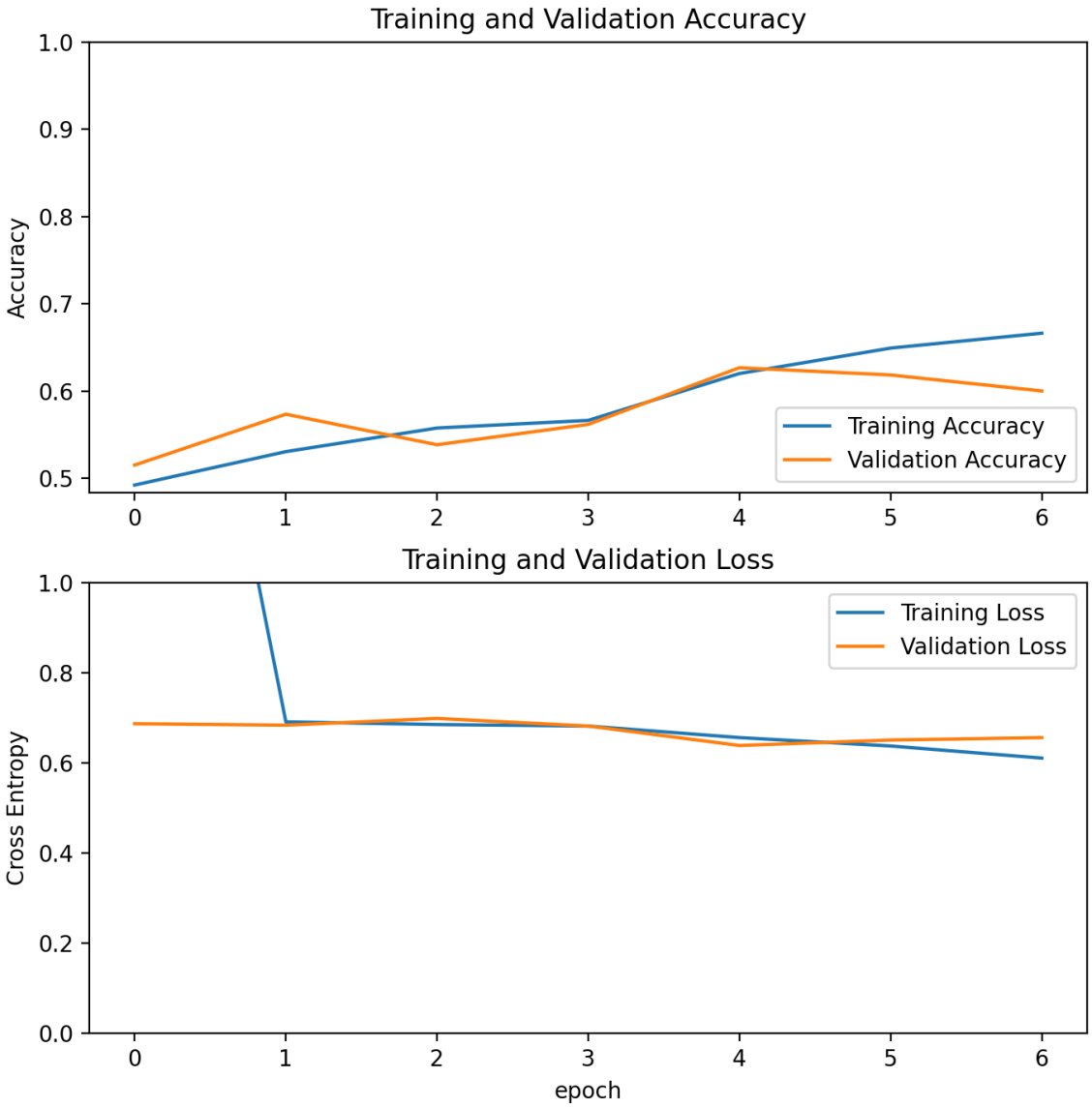

Figure 4. Training and validation accuracy for AlexNet model..

The accuracy for the train and validation set never reached 70%. It is possible that the model was overfitting as around the 4 epoch, the validation accuracy diverges in a downward trend opposite to the training accuracy.

10.3 VG19 Diagnostics and Readouts

10.3.1 Model Summary

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image (InputLayer)          [(None, 227, 227, 3)]     0

 sequential (Sequential)     (None, 227, 227, 3)       0

 tf.__operators__.getitem (  (None, 227, 227, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 227, 227, 3)       0
 )

 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 flatten_1 (Flatten)         (None, 512)               0

 dense_4 (Dense)             (None, 1000)              513000

 dropout_3 (Dropout)         (None, 1000)              0

 dense_5 (Dense)             (None, 1)                 1001

=================================================================
Total params: 20538385 (78.35 MB)
Trainable params: 514001 (1.96 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

Figure 5. Model summary for the VG19 transfer learning

10.3.2 Model History

```
Epoch 1/10
75/75 [==============================] - 266s 4s/step - loss: 0.7473 - accuracy: 0.8288 - val_loss: 0.3745 - val_accuracy: 0.91
50
Epoch 2/10
75/75 [==============================] - 261s 3s/step - loss: 0.4055 - accuracy: 0.8696 - val_loss: 0.3784 - val_accuracy: 0.88
00
Epoch 3/10
75/75 [==============================] - 248s 3s/step - loss: 0.2787 - accuracy: 0.8946 - val_loss: 0.2711 - val_accuracy: 0.90
67
Epoch 4/10
75/75 [==============================] - 248s 3s/step - loss: 0.2329 - accuracy: 0.9025 - val_loss: 0.2644 - val_accuracy: 0.91
50
Epoch 5/10
75/75 [==============================] - 255s 3s/step - loss: 0.2235 - accuracy: 0.9087 - val_loss: 0.2272 - val_accuracy: 0.91
17
Epoch 6/10
75/75 [==============================] - 250s 3s/step - loss: 0.1997 - accuracy: 0.9204 - val_loss: 0.3216 - val_accuracy: 0.89
00
Epoch 7/10
75/75 [==============================] - 249s 3s/step - loss: 0.1829 - accuracy: 0.9250 - val_loss: 0.3039 - val_accuracy: 0.90
00
```

Figure 6. Model history for VG19

10.3.3 Training and Validation Accuracy

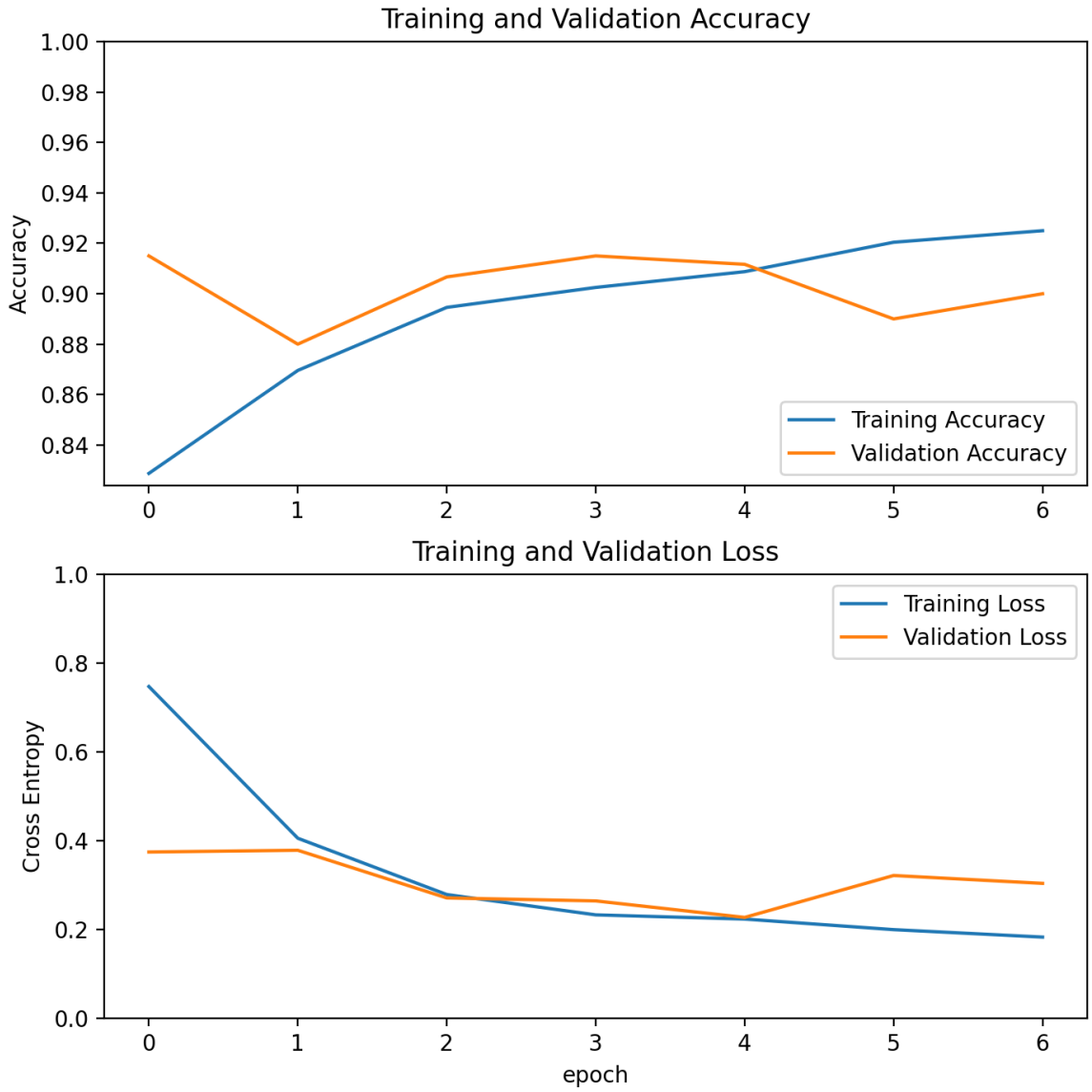

Figure 7. Training and validation accuracy for VG19 model

Comparing this to the AlexNet readouts, this is a huge improvement already. There still may be some overfitting within the model as around the 4th epoch there is a divergence in accuracy, but the validation accuracy does go back up after the 5th epoch.

11 Python Code

For best viewing, please refer to the included Jupyter notebook.

Administrative tasks

In [ ]:
```
#import sys
#!{sys.executable} -m pip install tensorflow
```

In [ ]:
```
# Admin packages
import pandoc
import numpy as np
import os
import pandas as pd
import pathlib

# Modeling packages for Neural Networks
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.preprocessing import image_dataset_from_directory
from collections import defaultdict

# Logistic Regression Code
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, classification_repo
from sklearn.preprocessing import StandardScaler, LabelEncoder

# EDA Packages
import matplotlib.pyplot as plt
import cv2
from plotnine import *
import plotnine
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
# Laptop
#train_dir= r'C:\Users\emili\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final
#test_dir= r'C:\Users\emili\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final P

# Desktop
train_dir= r'C:\Users\efv\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final Pro
test_dir= r'C:\Users\efv\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final Proj
```

Exploratory Data Analysis

EDA - Class Visualization

In [ ]:
```
# Data visualization with plotnine
from plotnine import *
import plotnine
```

In [ ]:
```
# Class visualization
```

```
# Function to count classes in a directory
def count_classes(directory):
    class_counts = {}
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            class_counts[class_name] = len(os.listdir(class_path))
    return class_counts

# Count classes in the training directory
train_counts = count_classes(train_dir)
test_counts = count_classes(test_dir)

# Create data frames for plotnine
train_data = pd.DataFrame({'Class': list(train_counts.keys()), 'Count': list(train_cou
test_data = pd.DataFrame({'Class': list(test_counts.keys()), 'Count': list(test_counts

# Combine data frames
combined_data = pd.concat([train_data, test_data], ignore_index=True)

# Create a bar chart with the legend at the bottom and without legend title
chart = (
    ggplot(combined_data, aes(x='Class', y='Count', fill='Dataset')) +
    geom_bar(stat='identity', position='dodge', color='black') +
    geom_text(aes(label='Count'), position=position_dodge(width=0.9), va='bottom') +
    labs(title='Class Distribution in Training and Test Data', x='Class', y='Count') +
    theme_bw() +
    theme(legend_position='bottom', legend_title=element_blank())
)

# Show the chart
print(chart)
```

EDA - Scatterplot of dimensions

In [ ]:

```
import os
import pandas as pd
from PIL import Image

def get_image_dimensions(base_dir):
    dimensions_list = []

    # Iterate through folders in the base directory
    for folder_name in os.listdir(base_dir):
        folder_path = os.path.join(base_dir, folder_name)

        # Check if the item in the base directory is a folder
        if os.path.isdir(folder_path):
            # Iterate through subfolders in the current folder
            for subfolder_name in os.listdir(folder_path):
                subfolder_path = os.path.join(folder_path, subfolder_name)

                # Check if the item in the current folder is a subfolder
                if os.path.isdir(subfolder_path):
                    # Iterate through files in the current subfolder
                    for filename in os.listdir(subfolder_path):
                        # Check if the file has a valid image extension
                        if filename.endswith(('.jpg', '.jpeg', '.png', '.gif')):
```

```
                            file_path = os.path.join(subfolder_path, filename)
                            with Image.open(file_path) as img:
                                # Get image dimensions
                                width, height = img.size
                                # Append information to the dimensions_list
                                dimensions_list.append({
                                    'folder': folder_name,
                                    'subfolder': subfolder_name,
                                    'filename': filename,
                                    'width': width,
                                    'height': height
                                })

    # Convert the list of dictionaries to a Pandas DataFrame
    return pd.DataFrame(dimensions_list)

# Base directory
base_dir = r'C:\Users\efv\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final Pro

# Get image dimensions and create a dataframe
image_dimensions_df = get_image_dimensions(base_dir)

# Print the dataframe
print(image_dimensions_df)
```

In [ ]:
```
# Create a facet scatterplot using plotnine
scatterplot = (
    ggplot(image_dimensions_df, aes(x='width', y='height'))
    + geom_point()
    + facet_grid('folder ~ subfolder')
    + labs(title='Image Dimensions of the Train and Test Datasets', x='Width', y='Heig
    theme_bw()
)

# Display the plot
print(scatterplot)
```

In [ ]:
```
# Function to read and preprocess images
def read_and_preprocess_image(file_path):
    img = cv2.imread(file_path)  # Read image using OpenCV
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
    return img

# Load training data into a DataFrame
train_data = []
for label in os.listdir(train_dir):
    label_dir = os.path.join(train_dir, label)
    if os.path.isdir(label_dir):
        for file_name in os.listdir(label_dir):
            file_path = os.path.join(label_dir, file_name)
            label_data = {"Label": label, "Dataset": "train", "Image": read_and_preprc
            train_data.append(label_data)

# Load test data into a DataFrame
test_data = []
for label in os.listdir(test_dir):
    label_dir = os.path.join(test_dir, label)
    if os.path.isdir(label_dir):
        for file_name in os.listdir(label_dir):
```

```
            file_path = os.path.join(label_dir, file_name)
            label_data = {"Label": label, "Dataset": "test", "Image": read_and_preproc
            test_data.append(label_data)

# Combine training and test data into a single DataFrame
#training_data = pd.DataFrame(train_data + test_data)
training_data = pd.DataFrame(train_data)
test_data = pd.DataFrame(test_data)

# Display the DataFrame
print(training_data)
```

In [ ]:
```
# Verify dimensions are 299x299 images read in for RGB values
first_row_image = training_data['Image'][0]
length_of_first_row = len(first_row_image)

print("Length of the first row:", length_of_first_row)

training_data['Image'][0].shape
```

In [ ]:
```
# Used for presentation to help visualize what a pixel is.
[
  [
    [223, 233, 235],  # Pixel at position (0, 0) with RGB values [223, 233, 235]
    [220, 225, 234],  # Pixel at position (0, 1) with RGB values [220, 225, 234]
    [243, 242, 249],  # Pixel at position (0, 2) with RGB values [243, 242, 249]
    # ... more pixels in the first row
  ],
  [
    [243, 242, 249],  # Pixel at position (1, 0) with RGB values [243, 242, 249]
    [220, 225, 234],   # Pixel at position (1, 1) with RGB values [220, 225, 234]
    [223, 233, 235],   # Pixel at position (1, 2) with RGB values [223, 233, 235]
    # ... more pixels in the second row
  ],
  # ... more rows
]
```

Splitting a single image to 3 channels

In [ ]:
```
# Splitting a single image into 3 diffrent layers

# Load the image
image_path = r'C:\Users\efv\OneDrive - Cal State Fullerton\2. Classes\Math 533\Final P

image = cv2.imread(image_path)

# Check if the image is loaded successfully
if image is None:
    print(f"Error: Unable to load image from {image_path}")
    exit()

# Split the image into its channels
b, g, r = cv2.split(image)

# Create a 2x2 subplot grid
#plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
plt.title('Original')
plt.axis('off')

#plt.subplot(2, 2, 2)
plt.imshow(b, cmap='gray')
plt.title('Blue Channel')
plt.axis('off')

#plt.subplot(2, 2, 3)
plt.imshow(g, cmap='gray')
plt.title('Green Channel')
plt.axis('off')

#plt.subplot(2, 2, 4)
plt.imshow(r, cmap='gray')
plt.title('Red Channel')
plt.axis('off')

plt.show()
```

RGB Line Graphs histograms - SKIP DURING RUNS

In [ ]:

```
# Train set, hotdog class
hotdog_test_data = training_data[(training_data['Label'] == 'hotdog') & (training_data

# Extract the 'Image' column from the filtered DataFrame
images = hotdog_test_data['Image']

# Function to create line graph for color histograms
def create_color_line_graph(images):
    # Initialize an array to store pixel values for each color channel
    red_values = []
    green_values = []
    blue_values = []

    # Extract pixel values for each color channel from all images
    for image in images:
        red_values.extend(image[:, :, 0].flatten())
        green_values.extend(image[:, :, 1].flatten())
        blue_values.extend(image[:, :, 2].flatten())

    # Calculate histograms for each color channel
    red_hist, _ = np.histogram(red_values, bins=100, range=(0, 256))
    green_hist, _ = np.histogram(green_values, bins=100, range=(0, 256))
    blue_hist, _ = np.histogram(blue_values, bins=100, range=(0, 256))

    # Plot line graphs for each color channel
    plt.figure(figsize=(10, 4))

    plt.plot(red_hist, color='red', label='Red')
    plt.plot(green_hist, color='green', label='Green')
    plt.plot(blue_hist, color='blue', label='Blue')

    plt.title('Color Histograms')
    plt.xlabel('Bin number')
    plt.ylabel('# of pixels')
    plt.legend()
```

```
    plt.show()

# Run the function on our image set
create_color_line_graph(images)
```

In [ ]:
```
# Train set, nothotdog class

hotdog_test_data = training_data[(training_data['Label'] == 'nothotdog') & (training_d

# Extract the 'Image' column from the filtered DataFrame
images = hotdog_test_data['Image']

# Function to create line graph for color histograms
def create_color_line_graph(images):
    # Initialize an array to store pixel values for each color channel
    red_values = []
    green_values = []
    blue_values = []

    # Extract pixel values for each color channel from all images
    for image in images:
        red_values.extend(image[:, :, 0].flatten())
        green_values.extend(image[:, :, 1].flatten())
        blue_values.extend(image[:, :, 2].flatten())

    # Calculate histograms for each color channel
    red_hist, _ = np.histogram(red_values, bins=100, range=(0, 256))
    green_hist, _ = np.histogram(green_values, bins=100, range=(0, 256))
    blue_hist, _ = np.histogram(blue_values, bins=100, range=(0, 256))

    # Plot line graphs for each color channel
    plt.figure(figsize=(10, 4))

    plt.plot(red_hist, color='red', label='Red')
    plt.plot(green_hist, color='green', label='Green')
    plt.plot(blue_hist, color='blue', label='Blue')

    plt.title('Color Histograms')
    plt.xlabel('Bin number')
    plt.ylabel('# of pixels')
    plt.legend()

    plt.show()

# Apply the function to the 'Image' column of the filtered DataFrame
create_color_line_graph(images)
```

EDA END

Modeling Begin

Logistic Regression

Data Prep

In [ ]:
```python
def calculate_average_rgb(data):
    """
    Calculate the average RGB value for each image and store it in a DataFrame.

    Parameters:
    - data: Pandas DataFrame containing image data

    Returns:
    - Pandas DataFrame with columns 'Label' and 'Avg_RGB'
    """
    # Create a list to store average RGB values for each image
    avg_rgb_list = []

    # Iterate through the data
    for index, row in data.iterrows():
        label = row['Label']
        image = row['Image']

        # Convert the image to a NumPy array
        image_array = np.array(image)

        # Calculate the average RGB value for the image
        avg_rgb = np.mean(image_array, axis=(0, 1))

        # Append the label and average RGB values to the list
        avg_rgb_list.append({'Label': label, 'Avg_RGB': avg_rgb})

    # Create a new DataFrame from the list
    avg_rgb_df = pd.DataFrame(avg_rgb_list)

    return avg_rgb_df
```

In [ ]:
```python
# Calculate avg rb for train and test set
avg_rgb_df_train = calculate_average_rgb(training_data)

avg_rgb_df_test = calculate_average_rgb(test_data)

print(avg_rgb_df_test)
```

In [ ]:
```python
# Extract RGB values into separate columns
avg_rgb_df_train[['R', 'G', 'B']] = pd.DataFrame(avg_rgb_df_train['Avg_RGB'].tolist(),

# Drop the original 'Avg_RGB' column as it is no longer needed
avg_rgb_df_train = avg_rgb_df_train.drop('Avg_RGB', axis=1)

# Again but for the test set

# Extract RGB values into separate columns
avg_rgb_df_test[['R', 'G', 'B']] = pd.DataFrame(avg_rgb_df_test['Avg_RGB'].tolist(), i

# Drop the original 'Avg_RGB' column as it is no longer needed
avg_rgb_df_test = avg_rgb_df_test.drop('Avg_RGB', axis=1)
```

```
# Display the modified DataFrame
print(avg_rgb_df_train)
```

Method 1 for LogReg - Dividing Predictors by 255 to scale between 0 and 1

In [ ]:

```
train_df = avg_rgb_df_train
test_df = avg_rgb_df_test

# Separate features and labels
X_train = train_df[['R', 'G', 'B']]
y_train = train_df['Label']
X_test = test_df[['R', 'G', 'B']]
y_test = test_df['Label']

# Encode labels into numerical values
label_encoder1 = LabelEncoder()
y_train_encoded = label_encoder1.fit_transform(y_train)
y_test_encoded = label_encoder1.transform(y_test)

# Rescale pixel counts to between 0 and 1
X_train_scaled = X_train/255
X_test_scaled = X_test/255

# Train the logistic regression model
model = LogisticRegression(random_state=69420)
model.fit(X_train_scaled, y_train_encoded)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model
zero_one_scale_accuracy = accuracy_score(y_test_encoded, y_pred)
report = classification_report(y_test_encoded, y_pred)

print(f"Accuracy: {zero_one_scale_accuracy:.2f}")
print("Classification Report:\n", report)

# Get the final weights
final_weights = model.coef_

# Print the final weights
print("Final Weights:")
for i, feature in enumerate(['R', 'G', 'B']):
    print(f"{feature}: {final_weights[0][i]:.4f}")

# Create a confusion matrix
cm1 = confusion_matrix(y_test_encoded, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder1.classes
plt.title('Logistic Regression with Rescale Between 0 and 1\n Accuracy = 54%')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Method 2 for LogReg - Standard Normal

In [ ]:

```
train_df = avg_rgb_df_train
test_df = avg_rgb_df_test

# Separate features and labels
X_train = train_df[['R', 'G', 'B']]
y_train = train_df['Label']
X_test = test_df[['R', 'G', 'B']]
y_test = test_df['Label']

# Encode labels into numerical values
label_encoder2 = LabelEncoder()
y_train_encoded = label_encoder2.fit_transform(y_train)
y_test_encoded = label_encoder2.transform(y_test)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the logistic regression model
model = LogisticRegression(random_state=69420)
model.fit(X_train_scaled, y_train_encoded)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model
std_norm_accuracy = accuracy_score(y_test_encoded, y_pred)
report = classification_report(y_test_encoded, y_pred)

print(f"Accuracy: {std_norm_accuracy:.2f}")
print("Classification Report:\n", report)

# Get the final weights
final_weights = model.coef_

# Print the final weights
print("Final Weights:")
for i, feature in enumerate(['R', 'G', 'B']):
    print(f"{feature}: {final_weights[0][i]:.4f}")


# Create a confusion matrix
cm2 = confusion_matrix(y_test_encoded, y_pred)


# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder2.classes
plt.title('Logistic Regression with Rescaled on Standard Normal\n Accuracy = 53%')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Method 3 - No Rescaling

In [ ]:
```
train_df = avg_rgb_df_train
test_df = avg_rgb_df_test

# Separate features and labels
X_train = train_df[['R', 'G', 'B']]
y_train = train_df['Label']
X_test = test_df[['R', 'G', 'B']]
y_test = test_df['Label']

# Encode labels into numerical values
label_encoder3 = LabelEncoder()
y_train_encoded = label_encoder3.fit_transform(y_train)
y_test_encoded = label_encoder3.transform(y_test)

# Multiply by 1 for no scale
X_train_scaled = X_train*1
X_test_scaled = X_test*1

# Train the logistic regression model
model = LogisticRegression(random_state=69420)
model.fit(X_train_scaled, y_train_encoded)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model
raw_predictors_accuracy = accuracy_score(y_test_encoded, y_pred)
report = classification_report(y_test_encoded, y_pred)

print(f"Accuracy: {raw_predictors_accuracy:.2f}")
print("Classification Report:\n", report)

# Get the final weights
final_weights = model.coef_

# Print the final weights
print("Final Weights:")
for i, feature in enumerate(['R', 'G', 'B']):
    print(f"{feature}: {final_weights[0][i]:.4f}")

# Create a confusion matrix
cm3 = confusion_matrix(y_test_encoded, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm3, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder3.classes
plt.title('Logistic Regression without Rescaling\n Accuracy = 52%')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Logistic Regression Results

In [ ]:
```
# Set up a 1x3 grid for subplots
fig, axes = plt.subplots(1, 3, figsize=(9, 3))

# Plot the first confusion matrix
sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder1.classes
axes[0].set_title('Model 1\nAccuracy = 54.03%')
axes[0].set_xlabel('Predicted Labels')
axes[0].set_ylabel('True Labels')

# Plot the second confusion matrix
sns.heatmap(cm2, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder2.classes
axes[1].set_title('Model 2\nAccuracy = 52.6%')
axes[1].set_xlabel('Predicted Labels')
axes[1].set_ylabel('True Labels')

# Plot the third confusion matrix
sns.heatmap(cm3, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder3.classes
axes[2].set_title('Model 3\nAccuracy = 52.5%')
axes[2].set_xlabel('Predicted Labels')
axes[2].set_ylabel('True Labels')

# Adjust layout to prevent clipping of titles
plt.tight_layout()

# Show the plots
plt.show()
```

Neural Network

In [ ]:
```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    # Restrict TensorFlow to only use the first GPU
    try:
        tf.config.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.list_logical_devices('GPU')
    except RuntimeError as e:
        # Visible devices must be set before GPUs have been initialized
        print(e)
```

In [ ]:
```
BATCH_SIZE = 32
IMG_SIZE = (227, 227) # We want to resize images to 256x256
```

Image Split

In [ ]:
```
# Set train set for Neural Network
train_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             validation_split=0.2,
                                             subset="training",
                                             seed=42,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE)
```

In [ ]:
```
class_names = train_dataset.class_names
class_names
```

In [ ]:
```
# Set validation set for Neural Network
validation_dataset = image_dataset_from_directory(train_dir,
                                                  shuffle=True,
                                                  validation_split=0.2,
                                                  subset="validation",
                                                  seed=42,
                                                  batch_size=BATCH_SIZE,
                                                  image_size=IMG_SIZE)
```

In [ ]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

In [ ]:
```
# Create test data set for neural network
test_dataset = image_dataset_from_directory(test_dir,
                                                  batch_size=BATCH_SIZE,
                                                  image_size=IMG_SIZE,
                                                  )
```

In [ ]:
```
# Autotuning parameters

AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Data preprocessing

Data augmentation and Rescale

In [ ]:
```
# Code to augment photos.
# Images are flipped, rotated, and randomly zoomed in
data_augmentation = tf.keras.Sequential(
    [
        preprocessing.RandomFlip("horizontal"),
        preprocessing.RandomRotation(0.2),
        preprocessing.RandomZoom(0.3)
    ]
)
```

In [ ]:
```
for image, _ in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    first_image = image[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
```

```
        plt.imshow(augmented_image[0] / 255)
        plt.axis('off')
```

In [ ]:
```
rescale = preprocessing.Rescaling(1./255)
```

Predictive Modeling via CNN

AlexNet

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import layers

def AlexNet():
    # Define the input layer with shape (227, 227, 3) to match AlexNet's input size
    inputs = tf.keras.Input(shape=(227, 227, 3), name='image')

    '''
    Layer 1
    '''

    # First convolutional layer with 96 filters, kernel size (11, 11), and ReLU activa
    x = layers.Conv2D(96, (11, 11), strides=4, activation='relu')(inputs)
    # Max pooling layer with pool size (3, 3), strides 2
    x = layers.MaxPool2D((3, 3), strides=2)(x)

    '''
    Layer 2 followed by max pool
    '''
    # Second convolutional layer with 256 filters, kernel size (5, 5), padding 'same',
    x = layers.Conv2D(256, (5, 5), padding='same', activation='relu')(x)
    # Max pooling layer with pool size (3, 3), strides 2
    x = layers.MaxPool2D((3, 3), strides=2)(x)


    '''
    Layer 3, 4, 5 followed by max pool
    '''
    # Third convolutional layer with 384 filters, kernel size (3, 3), padding 'same',
    x = layers.Conv2D(384, (3, 3), padding='same', activation='relu')(x)
    # Fourth convolutional layer with 384 filters, kernel size (3, 3), padding 'same',
    x = layers.Conv2D(384, (3, 3), padding='same', activation='relu')(x)
    # Fifth convolutional layer with 256 filters, kernel size (3, 3), padding 'same',
    x = layers.Conv2D(256, (3, 3), padding='same', activation='relu')(x)

    # Max pooling layer with pool size (3, 3), strides 2
    x = layers.MaxPool2D((3, 3), strides=2)(x)
    # Flatten the output for the fully connected layers
    x = layers.Flatten()(x)

    '''
    Fully connnected layers
    '''
    # First fully connected layer with 4096 units and ReLU activation
    x = layers.Dense(4096, activation='relu')(x)
    # Dropout layer with dropout rate of 0.5
```

```
    x = layers.Dropout(0.5)(x)

    # Second fully connected layer with 4096 units and ReLU activation
    x = layers.Dense(4096, activation='relu')(x)
    # Dropout layer with dropout rate of 0.5
    x = layers.Dropout(0.5)(x)

    # Third fully connected layer with 1000 units and ReLU activation
    x = layers.Dense(1000, activation='relu')(x)
    # Dropout layer with dropout rate of 0.5
    x = layers.Dropout(0.5)(x)

    # Output layer with 1 unit and sigmoid activation for binary classification
    outputs = layers.Dense(1, activation='sigmoid')(x)

    # Create the model with defined inputs and outputs, and name 'AlexNet'
    model = tf.keras.Model(inputs, outputs, name='AlexNet')

    # Return the created model
    return model
```

In [ ]:
```
model_alexnet = AlexNet()
```

In [ ]:
```
model_alexnet.summary()
```

In [ ]:
```
# Compile the model
model_alexnet.compile(
    # Set the optimizer for training the model
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0003),
    # Set loss function to Binary Crossentropy as it is commonly used for binary class
    loss=tf.keras.losses.BinaryCrossentropy(),
    # Use accuracy as metric measured in training
    metrics=['accuracy']
)
```

In [ ]:
```
# Creating an EarlyStopping callback for a Keras model
callback = tf.keras.callbacks.EarlyStopping(
    # Monitoring the validation loss during training
    monitor='val_loss',
    # Number of epochs with no improvement after which training will be stopped
    patience=2,
    # Restores model weights from the epoch with the best monitored metric (val_loss)
    restore_best_weights=True
)
```

In [ ]:
```
# Training using fit method
history_alexnet = model_alexnet.fit(
    # Training dataset
    train_dataset,
    # Number of training epochs
    epochs=10,
    # Validation dataset to monitor model performance during training
    validation_data=validation_dataset,
    # List of callbacks to be applied during training, including EarlyStopping
    callbacks=[callback]
)
```

In [ ]:
```
print(history_alexnet)
```

In [ ]:
```
def plotLearningCurve(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.ylabel('Accuracy')
    plt.ylim([min(plt.ylim()),1])
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 1, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.ylabel('Cross Entropy')
    plt.ylim([0,1.0])
    plt.title('Training and Validation Loss')
    plt.xlabel('epoch')
    plt.show()
```

In [ ]:
```
plotLearningCurve(history_alexnet)
```

In [ ]:
```
#model_alexnet.evaluate(test_dataset)
```

Results for CNN

In [ ]:
```
# Predictions on the test dataset
predictions = model_alexnet.predict(test_dataset)

# Convert predictions to binary values (0 or 1)
binary_predictions = (predictions > 0.5).astype(int)

# Get true labels
true_labels = np.concatenate([y for x, y in test_dataset], axis=0)

# Calculate confusion matrix
conf_matrix = confusion_matrix(true_labels, binary_predictions)

# Display confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Hotdog', 'No
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Print classification report
```

```
print("Classification Report:")
print(classification_report(true_labels, binary_predictions))
```

Pretrained VG19 Utilizing Transfer Learning

In [ ]:
```
# Initialize the VGG19 model with pre-trained weights. This was trained originally on
base_model_vgg19 = tf.keras.applications.vgg19.VGG19(
    weights='imagenet',        # Specify the pre-trained weights for the model
    include_top=False,         # Exclude the fully connected layers at the top of the
    input_shape=(227, 227, 3)  # Define the input's height, width, channels
)
```

In [ ]:
```
# Freezing the weights of the VGG19 model to prevent further training
base_model_vgg19.trainable = False
```

In [ ]:
```
# Define input layer with shape (227, 227, 3) and name 'image'
inputs = tf.keras.Input(shape=(227, 227, 3), name='image')

# Augment the input images to enhance model generalization
x = data_augmentation(inputs)

# Preprocess the input using VGG19-specific preprocessing
# This step ensures that input data is compatible with the pre-trained VGG19 model
x = tf.keras.applications.vgg19.preprocess_input(x)

# Pass the preprocessed input through the VGG19 base model
# The base_model_vgg19 was initialized with pre-trained weights on ImageNet
x = base_model_vgg19(x)

# Utilize Global Average Pooling to reduce spatial dimensions
# This layer computes the average value for each feature map, reducing spatial dimensi
x = layers.GlobalAveragePooling2D()(x)

# Flatten the output for further processing
# Flattening converts the 3D tensor output to a 1D tensor, preparing it for dense laye
x = layers.Flatten()(x)

# Add a Dense layer with 1000 units and ReLU activation
# Introducing a dense layer adds non-linearity and complexity to the model
x = layers.Dense(1000, activation='relu')(x)

# Apply Dropout with a rate of 0.3 for regularization
x = layers.Dropout(0.3)(x)

# Output layer with a single unit
outputs = layers.Dense(1)(x)

# Create a new model with the specified input and output layers, and name it 'vgg19'
model_vgg19 = tf.keras.Model(inputs, outputs, name='vgg19')
```

In [ ]:
```
model_vgg19.summary()
```

In [ ]:
```
# Compile the VGG19 model
model_vgg19.compile(
    # Specify the optimizer for training the model
    optimizer=tf.keras.optimizers.Adam(),
```

```
    # Specify the loss function to measure the difference between predicted and true v
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    # Specify metrics to evaluate the performance of the model during training
    metrics=['accuracy']
)
```

In [ ]:
```
# Define an Early Stopping callback
callback = tf.keras.callbacks.EarlyStopping(
    # Specify the monitored metric for early stopping
    monitor='val_loss',
    # Specify the number of epochs with no improvement after which training will be st
    patience=2,
    # Specify whether to restore model weights from the epoch with the best value of t
    restore_best_weights=True
)
```

In [ ]:
```
history_vgg19 = model_vgg19.fit(train_dataset,
                    epochs=10,
                    validation_data=validation_dataset,
                    callbacks=[callback])
```

In [ ]:
```
plotLearningCurve(history_vgg19)
```

In [ ]:
```
# Save the model just to see how to save a model
model_vgg19.save('model_vgg19.h5')
```

In [ ]:
```
model_vgg19.evaluate(test_dataset)
```

In [ ]:
```
# Initialize storage
test_images = []
test_labels = []

for image, label in test_dataset:
    test_images.append(image)
    test_labels.append(label)

test_images = np.vstack(test_images)
test_labels = np.concatenate(test_labels)

# Use trained model to make predictions
predictions = model_vgg19.predict(test_images)
predicted_labels = tf.math.sigmoid(predictions).numpy() > 0.5  # Assuming it's a binar

# Create a confusion matrix
conf_matrix2 = confusion_matrix(test_labels, predicted_labels)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix2, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted N
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Display classification report
#print(classification_report(test_labels, predicted_labels))
```

In [ ]:
```
# Labels for chart
xlab_chart = ['Hot Dog', 'Not Hot-Dog']
ylab_chart = ['Hot Dog', 'Not Hot-Dog']

# Plot the confusion matrices side by side
fig, axes = plt.subplots(1, 2, figsize=(9, 3))

# Plot the AlexNet confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',  xticklabels=xlab_chart, y
axes[0].set_title('AlexNet Model\nAccuracy = 50.1%')
axes[0].set_xlabel('Predicted Labels')
axes[0].set_ylabel('True Labels')

# Plot the VGG19 confusion matrix
sns.heatmap(conf_matrix2, annot=True, fmt='d', cmap='Blues', xticklabels=xlab_chart, y
axes[1].set_title('VGG19 Model\nAccuracy = 92.5%')
axes[1].set_xlabel('Predicted Labels')
axes[1].set_ylabel('True Labels')

# Adjust layout to prevent clipping of titles
plt.tight_layout()

# Show the plots
plt.show()
```

Final Results Combined

In [ ]:
```
# Set up a 2x3 grid for subplots
fig, axes = plt.subplots(2, 3, figsize=(12, 7))

# Plot the logistic regression confusion matrices
sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues', xticklabels=['Hot Dog', 'Not Hot-D
axes[0, 0].set_title('Log Reg Model 1\nAccuracy = 54.03%')
axes[0, 0].set_xlabel('Predicted Labels')
axes[0, 0].set_ylabel('True Labels')

sns.heatmap(cm2, annot=True, fmt='d', cmap='Blues', xticklabels=['Hot Dog', 'Not Hot-D
axes[0, 1].set_title('Log Reg Model 2\nAccuracy = 52.6%')
axes[0, 1].set_xlabel('Predicted Labels')
axes[0, 1].set_ylabel('True Labels')

sns.heatmap(cm3, annot=True, fmt='d', cmap='Blues', xticklabels=['Hot Dog', 'Not Hot-D
axes[0, 2].set_title('Log Reg Model 3\nAccuracy = 52.5%')
axes[0, 2].set_xlabel('Predicted Labels')
axes[0, 2].set_ylabel('True Labels')

# Plot the neural network confusion matrices
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=xlab_chart, yt
axes[1, 0].set_title('AlexNet Model\nAccuracy = 50.1%')
axes[1, 0].set_xlabel('Predicted Labels')
axes[1, 0].set_ylabel('True Labels')

sns.heatmap(conf_matrix2, annot=True, fmt='d', cmap='Blues', xticklabels=xlab_chart, y
axes[1, 1].set_title('VGG19 Model\nAccuracy = 92.5%')
axes[1, 1].set_xlabel('Predicted Labels')
axes[1, 1].set_ylabel('True Labels')
```

```
# Hide the empty subplot in the last column
axes[1, 2].axis('off')

# Adjust layout to prevent clipping of titles
plt.tight_layout()

# Show the plots
plt.show()
```