Food Image Classification and Calorie Estimation

John Rodocker and Paul Lintzen
CSCI 413: Data Science
Colorado School of Mines
December 10, 2025

I. Introduction

The focus of our project is to interpret and properly classify images of food items then predict the caloric information for the pictured item. Our plan is to train a model on a large dataset of food images and save that model for use in a simple app. This app will allow users to submit a photo of any food item and get a classification and calorie estimate back.

II. Data Collection

A. Dataset Sources

There are two primary data sources used in this project. The first is the Food-101 dataset which is used to build our model for image classification. The original dataset comes from ETH Zurich, but as their downloads website for the dataset is no longer active we used a version of it published on Kaggle titled Food Images (Food-101). The second source is the USDA FoodData Central API, which was used to retrieve nutritional information for all of the foods present in Food-101. The third source is the Nutritionix API, used for supplementing information not found in the FoodData Central API.

B. Image Data Organization

Images are provided in a directory structure where each subfolder corresponds to a unique food class. This folder-based organization facilitates easy label assignment during preprocessing. For example:

- `images/pizza/` contains all pizza images
- `images/sushi/` contains all sushi images

Each subfolder contains 1000 different images of that food and in total we have 101 foods represented meaning 101,000 total images.

C. Nutrition Data Retrieval and Organization

For retrieving an estimate of the calorie count for each food, we will be using the FoodData Central API and supplementing it with the Nutritionix Natural Language API for dishes which don't have a USDA entry. The units of the calorie information will be kcal/100g unless we find a different metric to be more useful later on.

To retrieve the data from FoodData Central, the food names were first parsed into a space-separated format, then used as a query for the `/foods/search` API endpoint. This endpoint is used to search the provided database for entries which have a matching name, where the `Survey (FNDDS)` data type is used to search only through complete dishes (e.g. "foie gras") rather than base ingredients (e.g. "beef"). Limiting results to 10 for brevity, the calorie counts for the results are averaged to come up with a relatively accurate estimate for the calorie count of the dish in general.

There are 8 foods which do not have listings in FoodData Central, for which we instead used the `/natural/nutrients` endpoint of the Nutritionix API to retrieve calorie counts. For consistency, the units for this are converted in kcal/100g to match the FoodData Central data. Since these calorie counts will only be linked to the food classification determined by the image processing section of the application, these are stored in a dictionary with the food name as the key and estimated kcal/100g as the value.

III. Data Preprocessing

A. Overview

Preprocessing ensures that the dataset is consistent, clean, and in a format that allows us to start training our model. The following steps were applied:

B. Resizing Images

All images were resized to 128×128 pixels to reduce computational complexity while retaining sufficient detail for classification. The original images were all 384×512, so this lead to considerable space savings and compute time reductions. This portion of the code was written by ChatGPT and the resizing was performed using Python's Pillow and tqdm library. Here is the basic process:

Listing 1: Pseudo code for resizing images

```
for each class folder in the images directory:
    create new folder in food-128 directory
    for each image in the class folder:
        Open image and convert to RGB
        Resize image to 128x128 using bilinear
            interpolation
        Save resized image in the food-128
```

We then validated that the images were successfully down-scaled by testing that each subdirectory in the new food-128 directory had 1000 jpg files and all the food categories were still present.

C. Train/Validation Split

To prepare the dataset for training and evaluation, each class folder was split into training, testing, and validation sets with an 70/15/15 ratio. The split preserves the directory structure:

- `food-128-split/train/pizza/`
- `food-128-split/test/pizza/`
- `food-128-split/val/pizza/`

The process for this is outlined in the following pseudo code:

Listing 2: Pseudo code for splitting data

```
for each class in food-128 folder:
    list all images in class folder
    shuffle images randomly
    split images into 70% train, 15% test, 15% val
    copy train images to /train/class
    copy test images to /test/class
    copy test images to /val/class
```

D. Converting to Machine Learning Format

At this point we have all our images successfully downscaled and split up, but they aren't yet in a format that we can train a model on. For this purpose we originally used PyTorch, which handles the conversion of the jpg image files to something more workable. We still kept PyTorch for our EDA and very basic first model but switched to TensorFlow for later models as it is a lot simpler to work with and allowed for greater flexibility. The conceptual idea of the conversion process is the same for both but there are some formatting differences. The basic process for PyTorch is as follows:

1) PyTorch uses the Pillow library to convert each image to RGB values and turn it into a 128x128x3 array, where the 3 represent the 3 color channels.
2) We apply a transform which converts from the PIL image to a PyTorch tensor (another type of multidimensional array).
3) Next, we normalize our color values to ensure similar scales, going from [0, 255] to [0.0, 1.0] or [-1, 1] (depending on specific model) for each channel.
4) The tensors are then grouped into batches by our dataloader, which returns a tuple of the batch images and batch labels.
5) The original text batch labels are converted to corresponding integer indices in alphabetical order and each converted image is then associated with one of these encoded labels
6) At this stage our data is in a 4D tensor formatted as (batch_size, channels, height, width) that we can then feed into our model

E. Challenges and Solutions

- **Missing / Inconsistent Entries in USDA Database:** Some food entries, mostly international dishes, did not have a result when queried on the USDA Food API so this needed to be supplemented with a secondary source. For this, we decided on the Nutritionix API as it in general provides decently accurate entries for lesser known foods. Additionally, some foods came back with hundreds of results, some related and some not, and in this case we decided to take the top 5 entries and average the calories across them. If this presents issues in accuracy later on, we will revisit this approach and propose a new solution.
- **Memory limitations / Compute Time:** The original images at full resolution took up ∼6GB of storage and also took forever to process or move around. By resizing all the images to 128x128 we cut the storage size down to ∼800 MB. Even at this size processing took a while with the down scaling taking roughly 30 minutes and the train test splitting another 20.
- **Converting Images to a Machine Learning format:** This seemed like it would be more difficult than it ended up being. By using the PyTorch Python package we were able to read in our downscaled and split data with relative ease and convert it into a format to feed into at model.

F. Summary

Through these preprocessing steps, we went from having raw images and lots of data at our disposal through an API to formatted 4D tensors for all our images and a dictionary mapping all our foods to a calorie estimate for a 100g portion.

IV. Exploratory Data Analysis

A. Numerical Summaries

Since we are working with image data, our numerical summaries are less obvious to choose. We can first look at some basic summary statistics to confirm our dataset is structured the way we think. We can ensure that the images are all 128x128 pixel and in RGB. Additionally, we can confirm that each class has 800 images as we expect (since we are using 80% of the total 1000 to train our model). Looping through our tensors we find that we have:

- 80,800 total tensors/images
- Each image is 3 channels, 128x128 pixels
- Each food class has 800 unique images

So we have confirmed that the structure is as we expected and can focus on some more interesting analysis in our visual summaries.

For the calorie estimates (kcal/100g) of each of the classes, we also calculated basic summary statistics:

- Mean: 219.405941
- Standard Deviation: 96.642517
- Minimum Value: 43.0
- Maximum Value: 462.0

B. Visual Summaries

Since we are examining pictures of 101 different types of food items, we will only create visualizations for a small subset of these for brevity. Our chosen visualizations are the average images and the average color channel values for each of our selected classes.

The 8 foods we chose to demonstrate this, pictured in Fig. 1, are each unique enough to have average images which are somewhat distinct from one another. Though not very useful on their own, these average images allow us to see the general patterns of color presence in a class of food in comparison to others. The average image for beef carpaccio in particular is interesting because of the presence of green in the center with

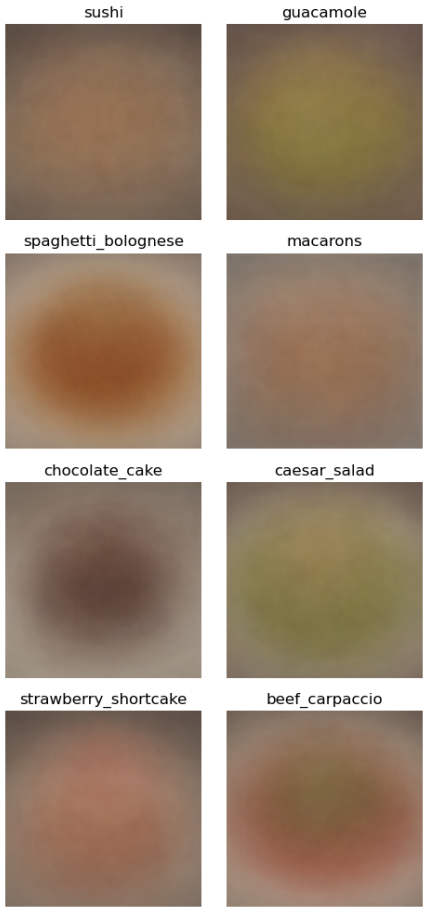


Fig. 1: Collection of average images for 8 selected foods

red surrounding it. This dish is often served as a plate of thinly sliced raw beef with a green garnish on top, so it is promising to see the average image reflect this. For the other selected classes we can also see some visual patterns reminiscent of the food which the images are of, though to a lesser degree.

The average images pictured in Fig. 1 were obtained by summing all the PyTorch tensors for the non-standardized training data of the chosen class, then dividing by the number of observations in the training data. We combine the tensors for each of the 8 chosen foods to create a pixel-wise mean of all training data for that class, which can then be rendered as an image that represents the average color per pixel in that class.

Next, we will look at the average color channel values for each of the same 8 foods, pictured in Fig. 2. This visualization was creating by finding the mean of the all pixels present in the average image for each of our 8 chosen classes, then creating a bar chart breaking down the specific color channel values for the average pixel.

In addition to this exploration of mean values, we also took a look at standard deviation in pixel values both within and across classes. The standard deviation in average color within each class is pictured in Fig. 3. This reveals that there is some

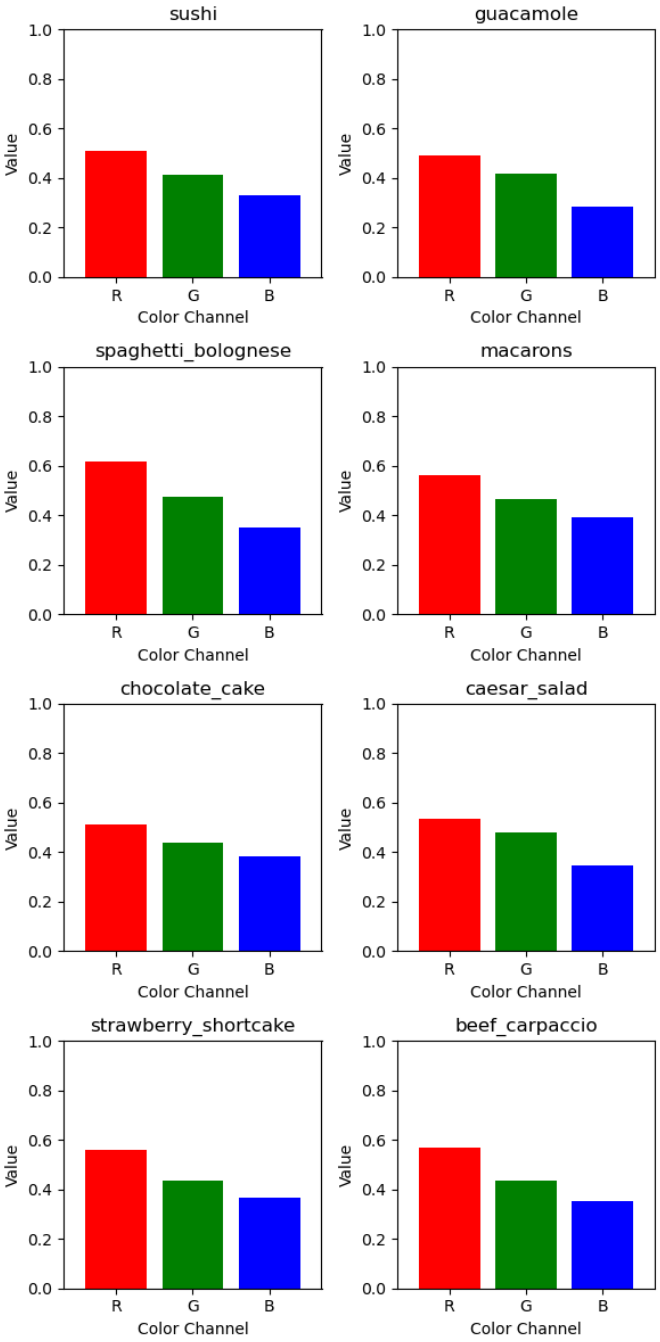


Fig. 2: Collection of average pixel color channel values for 8 selected foods

difference in the variation in color between our 8 selected classes. While the difference in standard deviation per class isn't major for any of these classes, with a sample of 800 images we can be relatively confident that some difference exists. This means that some food categories generally have more deviation in the colors pictured than others. It is important to note that this is averaging the pixel values of the entire image including background and plates so its not the most accurate way to look purely at the colors of the food but does

give us some idea.

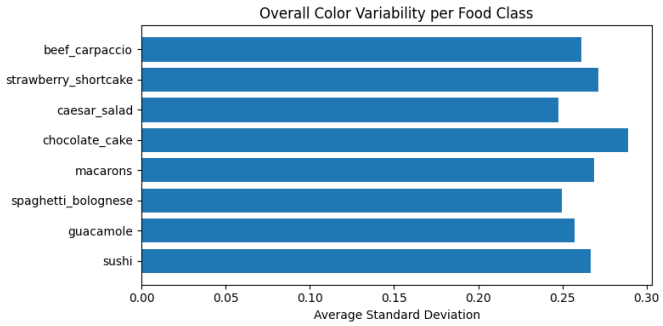


Fig. 3: Standard deviation in average pixel color between images of same food

By default, the returned image looks solid black, but if we divide each pixel's value by the max deviation we can massively increase our contrast revealing the image seen in Fig. 4. The areas of this figure that are most bright are those with the most variation while the darker areas are those with the least. This visualization reveals that most of our dishes seem to have a circular patch around them that is very similar across all the classes. This is likely the plate that each dish is on. Since a lot of dishes are on some sort of white plate we get this area with much less deviation in average color than the rest of the image. Then, in the center we have a bright spot where the dishes usually sit. It doesn't come across perfectly in this document but the outer edges are quite a bit dimmer than the central white patch, indicating that while we do have some variation it is not as great as the variation in the center. This makes sense as the background does vary from image to image but not nearly as much as the center of the image where our food usually sits. Overall, this result makes a lot of sense considering what we know about the data from visual inspection.

Fig. 4: Standard deviation of each pixel across classes

For visualizing the calorie estimates we assigned to each of the classes, we created a bar chart which shows the calorie counts of all foods at once, pictured in Fig. 5. This shows us just how large the range in calories is across our food

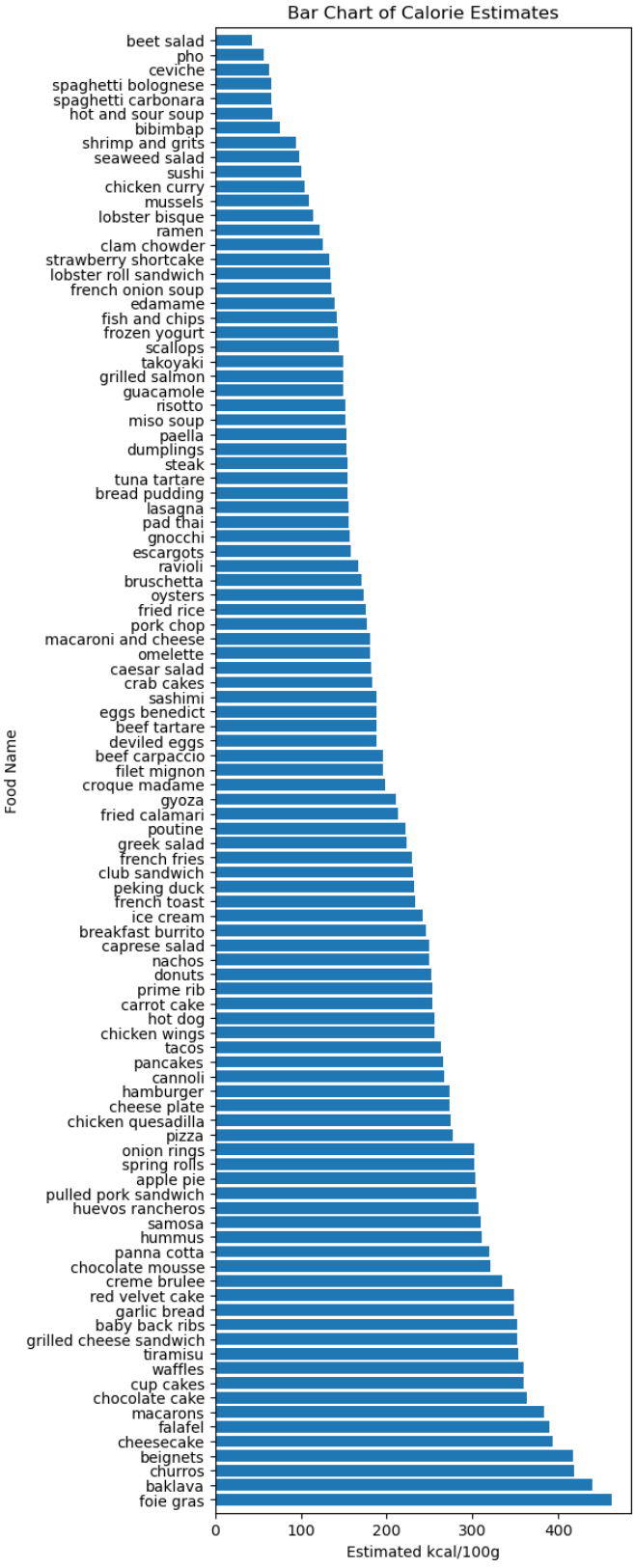


Fig. 5: Bar chart of the estimated kcal/100g for each of the 101 foods

categories. From our earlier numerical analysis we know we have foods as low as 43 kcal/100g and up to 462 kcal/100g.

V. Model Development

Since this project relies most on image recognition, the two models we chose as candidates for use are a Convolutional Neural Network (CNN) and a Vision Transformer (ViT), as both of these are used frequently for this task.

A. Convolutional Neural Network

CNN's were our first choice of model as they are widely used for image classification tasks and have been around for a relatively long time, first being developed in the 1980s, meaning there is a lot of information out there about tuning and training them. In choosing a CNN for this project, the main consideration is the CNN architecture. Using the methods we were taught in class, we first went through six iterations of architectures with varying degrees of success.

B. Vision Transformer

Vision transformers on the other hand are much more cutting edge, first being introduced in 2020. These generally perform better than CNNs in computer vision benchmarks but are also more complex to train and require much more data. To keep things reasonable, we used a pretrained vision transformer model from Tensor Flow hub and fine-tuned it to our dataset. This way we avoid lots of extra training time and are able to build on a solid foundation. The pretrained model we chose is VIT-B8-Classification created by Sayak Paul. He has a ton of open source models to choose from and many models trained on the ImageNet1k dataset which contains over 1 million unique images across 1000 distinct categories. Training this kind of model from scratch would take us weeks and is really not feasible with the resources we have available. Even just feeding our data into this pretrained model and getting predictions values took around 2 hours on our laptops. To put this into perspective, the entire training process and evaluation of our most complex CNN took around 45 minutes so this a considerable jump in complexity.

VI. Results and Evaluation

A. CNN Results

To evaluate our CNN we will start with our initial model and see how the results changed over time. Our first model implemented 4 'blocks', each with a convolutional layer followed by a max pooling layer to reduce the dimensionality. The convolutional layers had 32 nodes in the first, 64 in the next, then 128, followed by 256, and finally 101 in our output layer to determine our best guess as to what class the input image is.

While not completely useless our initial model was not great. We see telltale signs of overfitting with our train accuracy getting all the way to 70% while our test accuracy peaked around 15%. This inspired our first set of changes to get to architecture 2. Our second architecture uses batch normalization layers in each block, then a single dropout

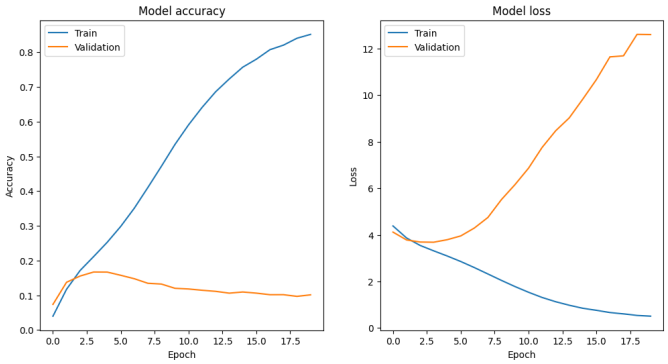


Fig. 6: Initial Architecture Results

layer before the final classification layer. The results with 10 epochs are shown below: We see a pretty big improvement

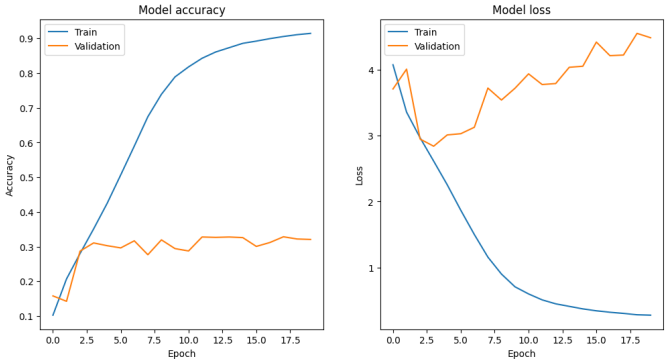


Fig. 7: Architecture 2 Results

in our validation accuracy in this, now peaking just around 30%. Interestingly, the dropout layers also mean that our accuracy transition between epochs is much less smooth. Our first thought upon seeing these results was to attempt to just run the model for 20 epochs to see if the validation accuracy climbed any further, but ultimately it continued to oscillate at around 30%. After more iterations of tuning and architecture change, we ended up with our next model that actually showed some improvement; architecture 5. This model architecture follows a VGG-Net style with use of a global average pooling layer at the beginning of the classification head. The goal of introducing this layer instead of creating a dense layer of all 128 32×32 filters is to dramatically reduce the number of trainable parameters in the model. This results in greatly reduced training time and seemed to result in less overfitting overall.

Now our validation accuracy was staying a lot more in line with the test accuracy throughout epochs, indicating that we were doing a good job extracting actual trends in the underlying foods rather than overfitting to specific features of the training data. With this architecture we moved up to a new peak of approximately 40% accuracy on our validation data. At this point we started to struggle to find more improvements. No changes we made seemed to result in meaningful performance

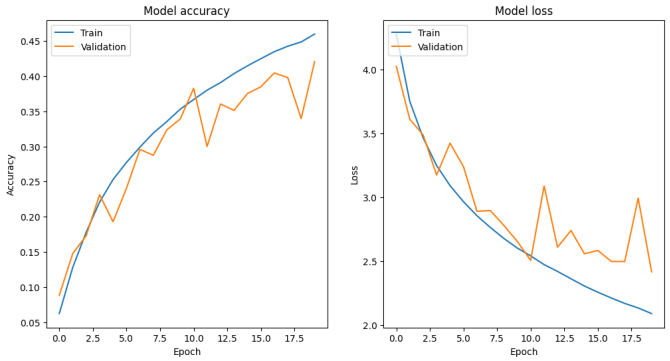


Fig. 8: Architecture 5 Results

improvements, so we began looking at other options. After some initial tuning with a new architecture, we got some very promising results. This architecture uses MobileNetV2, a powerful but complex CNN architecture, as the base, with our own layers of tuning on top of it.

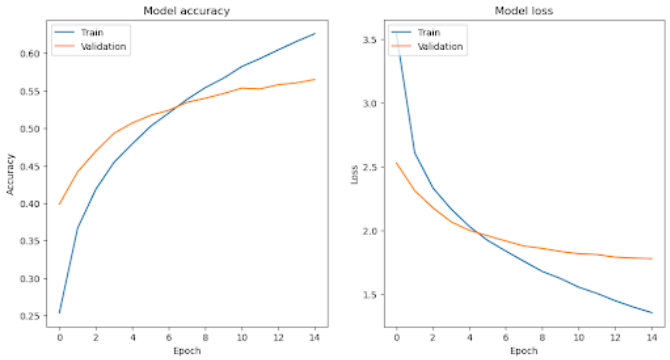


Fig. 9: Architecture 7.1 Results

In 14 epochs we got up to a validation accuracy of about 55%, another 15% jump from our last architecture. With this early success, we started experimenting with hyperparameters and increasing the number of epochs to see how far we could push this new model. Eventually though we again hit what felt like a softcap.

We see that even after tuning and increasing the epochs we got maybe another 2-3% out but nothing groundbreaking. At this point we're reasonably happy with our model. While 57% still isn't as good as we had hoped, it is significantly better than our first architecture and is good enough to where we can start looking at other ways of improving the model.

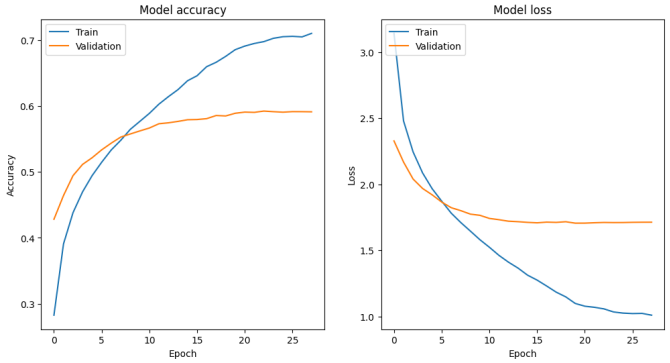


Fig. 10: Architecture 7.2 Results

B. Final CNN Evaluation

To do some further evaluation of this model we printed the classification report and confusion matrix to get an idea of if there were specific classes that we were performing significantly worse on. There was actually a fairly large range in our performance for different classes, with foods that have more distinct color and shapes usually performing better. Our model generally performed worse on foods more neutral in color or with inconsistent shape which makes sense. The table below includes our classification report for our final model to show how we're performing on specific classes. Overall, on the validation data we're getting an average accuracy, f1-score, precision, and recall of 59%. While this is far from ideal, it is passable especially considering how much we downscaled the images.

TABLE I: Classification Report for Food101

Class	Precision	Recall	F1-score	Support
apple_pie	0.33	0.31	0.32	150
baby_back_ribs	0.63	0.69	0.66	150
baklava	0.62	0.65	0.63	150
beef_carpaccio	0.60	0.62	0.61	150
beef_tartare	0.47	0.45	0.46	150
beet_salad	0.50	0.55	0.53	150
beignets	0.73	0.82	0.77	150
bibimbap	0.73	0.77	0.75	150
bread_pudding	0.34	0.38	0.36	150
breakfast_burrito	0.43	0.44	0.44	150
bruschetta	0.45	0.39	0.42	150
caesar_salad	0.59	0.67	0.63	150
cannoli	0.62	0.64	0.63	150
caprese_salad	0.51	0.49	0.50	150
carrot_cake	0.68	0.56	0.61	150
ceviche	0.39	0.36	0.38	150
cheese_plate	0.51	0.49	0.50	150
cheesecake	0.42	0.37	0.39	150
chicken_curry	0.47	0.39	0.42	150
chicken_quesadilla	0.60	0.56	0.58	150
chicken_wings	0.62	0.72	0.67	150
chocolate_cake	0.48	0.53	0.50	150
chocolate_mousse	0.34	0.42	0.38	150
churros	0.70	0.74	0.72	150
clam_chowder	0.72	0.74	0.73	150
club_sandwich	0.69	0.63	0.66	150
crab_cakes	0.43	0.38	0.40	150
creme_brulee	0.71	0.71	0.71	150
croque_madame	0.70	0.75	0.72	150
cup_cakes	0.69	0.71	0.70	150
deviled_eggs	0.75	0.75	0.75	150
donuts	0.58	0.55	0.57	150
dumplings	0.80	0.77	0.79	150
edamame	0.95	0.94	0.94	150
eggs_benedict	0.74	0.71	0.72	150
escargots	0.64	0.59	0.62	150
falafel	0.51	0.45	0.48	150
filet_mignon	0.46	0.37	0.41	150
fish_and_chips	0.64	0.67	0.65	150
foie_gras	0.30	0.30	0.30	150
french_fries	0.78	0.82	0.80	150
french_onion_soup	0.71	0.57	0.63	150
french_toast	0.50	0.51	0.50	150
fried_calamari	0.58	0.60	0.59	150
fried_rice	0.64	0.69	0.66	150
frozen_yogurt	0.76	0.72	0.74	150
garlic_bread	0.51	0.49	0.50	150
gnocchi	0.44	0.49	0.47	150
greek_salad	0.56	0.58	0.57	150
grilled_cheese_sandwich	0.53	0.41	0.46	150
grilled_salmon	0.42	0.34	0.38	150
guacamole	0.71	0.71	0.71	150
gyoza	0.58	0.65	0.62	150

Class	Precision	Recall	F1-score	Support
hamburger	0.60	0.57	0.59	150
hot_and_sour_soup	0.82	0.85	0.83	150
hot_dog	0.64	0.67	0.66	150
huevos_rancheros	0.35	0.31	0.33	150
hummus	0.41	0.39	0.40	150
ice_cream	0.49	0.58	0.53	150
lasagna	0.55	0.48	0.51	150
lobster_bisque	0.68	0.74	0.71	150
lobster_roll_sandwich	0.67	0.63	0.65	150
macaroni_and_cheese	0.49	0.55	0.52	150
macarons	0.78	0.80	0.79	150
miso_soup	0.88	0.81	0.85	150
mussels	0.72	0.83	0.77	150
nachos	0.51	0.56	0.53	150
omelette	0.41	0.36	0.38	150
onion_rings	0.68	0.82	0.75	150
oysters	0.81	0.77	0.79	150
pad_thai	0.70	0.67	0.68	150
paella	0.61	0.67	0.64	150
pancakes	0.60	0.61	0.60	150
panna_cotta	0.45	0.52	0.48	150
peking_duck	0.60	0.67	0.63	150
pho	0.86	0.84	0.85	150
pizza	0.70	0.81	0.75	150
pork_chop	0.31	0.30	0.30	150
poutine	0.73	0.69	0.71	150
prime_rib	0.65	0.62	0.64	150
pulled_pork_sandwich	0.56	0.55	0.56	150
ramen	0.68	0.63	0.66	150
ravioli	0.45	0.30	0.36	150
red_velvet_cake	0.68	0.75	0.72	150
risotto	0.48	0.46	0.47	150
samosa	0.60	0.58	0.59	150
sashimi	0.74	0.74	0.74	150
scallops	0.44	0.43	0.43	150
seaweed_salad	0.74	0.79	0.77	150
shrimp_and_grits	0.46	0.49	0.48	150
spaghetti_bolognese	0.73	0.75	0.74	150
spaghetti_carbonara	0.79	0.87	0.83	150
spring_rolls	0.65	0.63	0.64	150
steak	0.34	0.31	0.32	150
strawberry_shortcake	0.53	0.57	0.55	150
sushi	0.53	0.53	0.53	150
tacos	0.36	0.36	0.36	150
takoyaki	0.57	0.63	0.60	150
tiramisu	0.55	0.47	0.51	150
tuna_tartare	0.37	0.34	0.35	150
waffles	0.61	0.61	0.61	150

C. Vision Transformer Results

Though our vision transformer setup is technically functioning, we do not have the computing power to adequately use it. Having loaded all of our data correctly and setting up the pre-trained transformer, it produced a predicted runtime of 3 hours and crashed 15 minutes in. A fully trained vision transformer would likely be another substantial jump in performance over our final CNN architecture, but this will have to wait until another time.

VII. Ethical Considerations

A. Privacy

Since our project deals with user uploaded images, data privacy will be one of the main concerns. We will ensure that all user uploaded images are fully anonymous and are deleted as soon as our model has been run. To ensure full transparency we will include a popup in our flask application that explains to users what exactly will happen when they upload an image and how it will be used in our model. We will explain that it will only be retained for the duration of their use and will not be stored beyond that.

Another aspect of data privacy we need to consider for this project is the acquisition of data used for training it. The license for the dataset we used for training the current version of our model stipulates that any use beyond scientific fair use must be negotiated with the respective owners of each image, which would become a challenge if this were to be deployed as a production model. For our proof of concept use of these images this doesn't present an issue, but the implications of scaling up to a larger model is worth considering. A method commonly used today for vast data acquisition is web scraping, and in this case would give us access to a massively larger number of images of foods both included in and excluded from our list of 101 dishes. However, this method very often treads over any copyrights and licenses associated with collected image data, and for those it doesn't it still lacks the proper usage permission from those who post it.

B. Bias

To mitigate bias in our model we used a balanced training set and normalized all of our pixel data. However our model is incredibly biased to the training data we used. Since we only trained on 101 foods, that is all our model knows, so if a user uploads an image of a food that our model has ever seen it will still classify it as one of its known 101 foods. While the Food-101 dataset contains a relatively diverse cast of foods, it has a heavy skew towards Western (meaning American and Western European) foods. For a rough breakdown, approximately 68% of the dataset is Western foods, 8% is Japanese, 5% is Mexican/Latin American, 5% is Chinese/East Asian, 2% is Thai/Vietnamese, 2% is Indian, 2% is Middle Eastern, and the remaining 8% is anything else. This means that for non-western stakeholders our current model would be close to useless and only telling them what western food their dish looks most similar to. This is obviously not an ideal solution, and in a real world production scenario we would want to use a much more diverse dataset that adequately represents global cuisines. To further improve the model we could even consider having models fine tuned for different regions and be automatically chosen based on location data; although this again brings up questions of data privacy and generalization.

C. Fairness

Fairness was already briefly touched on above, but it will be stated gain here for clarity. For our current model, there is a distinct skew towards Western foods and as a result it will perform significantly worse for foods from other cultures. For a production model this is a real issue because we would be over-representing the majority western culture while under-representing everything else. From a global perspective this means our app would be close to useless to a larger global audience and possibly downright disrespectful as it would just spit out whatever western dish looks most similar. For the purposes of this project as a proof of concept, this is acceptable as it is not realistic to collect enough data to provide a model that works equally well across all foods. As long as we are aware of this and don't try to deploy this as a production model, it is just a necessary sacrifice to get the model working in a limited time frame and have it train in a reasonable time on our hardware.

VIII. Conclusion

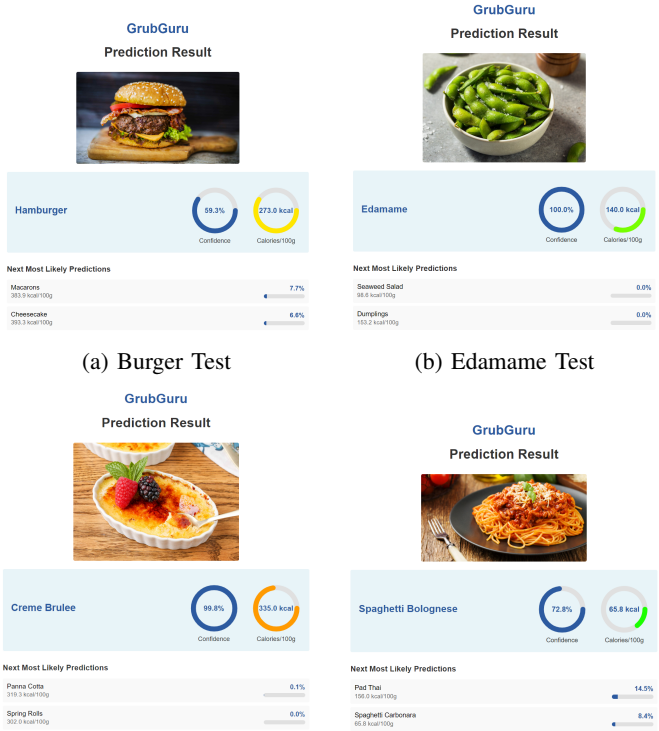
The goal of this project was to train a model to accurately classify different varieties of food and supply a calorie estimate for the label determined. In doing so, we ingested 1,000 images each of 101 different foods using the Food-101 dataset, and queried the FoodData Central API and Nutritionix API to retrieve the accompanying caloric information. The different forms of models we used required slightly different preprocessing steps, but for all of them the images were scaled to a resolution of 128×128, using bilinear interpolation, to reduce the training time and amount of data that needed to be stored locally. This limited the potential performance of all our model iterations, but resulted in a more feasible prototype. Using these images, we trained three varieties of CNN model architectures with varying levels of accuracy, with our final decision being to use a MobileNetV2 architecture. Though a Vision Transformer would likely have resulted in higher classification accuracy, the computing requirements to do so were too prohibitive for the hardware we had access to.

Overall, we succeeded in creating a model that correctly classified images in our validation data set 59% of the time. While this is far from perfect, considering that we downscaled the images considerably and used a relatively simple model so we could actually train it in a reasonable time on the hardware we had available, this is a pretty decent score. With 101 unique categories 59% is considerably better than random guessing and it means our model did actually start to understand the data to some degree. Additionally, in many cases where our model predicted wrong it would have the correct prediction in second or third place meaning it had at least some understanding of what different foods looked like.

There are lots of options for future improvements to this model, but here are just a few. The most obvious way to improve our model is just to downscale less. By moving up to a resolution like 224×224 we could retain a lot more information from the images; though, it would be at the cost of performance as this would triple the amount of pixel data. Another option would be switching to a different model like a Vision Transformer. If we had access to more powerful computers and spent some more time ironing out the kinks, a Vision Transformer model would almost certainly outperform our current one. Alternatively, we could also switch to a different pretrained model like EfficientNetV2 which might slightly increase training time but would likely also improve accuracy. We could also experiment further with different data augmentation to ensure our model doesn't get overfit to some specific features of our training data. Since users won't take images all from the same angle and in the same lighting, it is important to account for that in our training. Additionally, having more images to train on would definitely improve the accuracy of our model and adding more categories would make it more applicable to wider audience even outside the western world. As a final improvement it would be super interesting to add portion estimates so we return customized calorie counts based on the size of the food in the image rather than just per unit count.

IX. More Information

For more information on our project or to try it out yourself check out our GitHub Repo. For a little sneak peak at our web application here are a few results with images we found online:



(a) Burger Test

(b) Edamame Test

(c) Creme Brulee Test

(d) Spaghetti Bolognese Test

Fig. 11: Sample Results on our website