What's in that Dish?

Vijay Muppalla (903030059) 50%
vmuppalla6@gatech.edu

Udechukwu Chinenye Adanna (903944961) 50%
cudechukwu3@gatech.edu

December 28, 2024

1 Introduction

Due to a global surge in disposable income and easier access to travel, people are experiencing new cuisines and cultures more than ever before [1]. The phenomenon of globalization has created opportunities for average citizens to experience food from around the world from the comfort of their homes. This gives rise to a unique problem. People have different taste preferences and dietary restrictions. So knowing the ingredients of a dish is very important. Currently many restaurants, small or large, do no provide detailed ingredient lists for their dishes. It can also be time consuming, tedious, and embarrassing to have to regularly ask about the contents of food in social settings. In the case of travel, there could also be language barriers that make this process harder.

In light of these issues our team has set out to develop and deploy a machine learning model that identifies dishes based on their picture and provides a list of common ingredients. In this paper, we will provide details about the model structure, metrics, and mobile deployment of our solution.

2 Previous Work

The most significant research was conducted by a group of researchers from the Massachusetts Institute of Technology, the Qatar Computing Research Institute, and the HBKU Universitat Politecnica de Catalunya. Their research was based on the Recipe 1M+ dataset, which contains more than thirteen million images. These images are associated with one million different recipes within the dataset. They did a form of multimodal learning by creating joint neural embeddings between images and recipes [2]. They used LSTMs to create word embeddings for the recipes while using VGG-16 and ResNet-50 for the image representation. There was one attempt to deploy a mobile interface that would use this model called 'Edible', but the project has since been abandoned [3].

A group of Stanford students also conducted similar research, where the goal was to obtain recipes from images of food. One unique aspect of their project was that they used a KNN (K-Nearest Neighbors) to produce recipe lists for a given image. They compared the distance of the input images' encoding to that of other images to determine the closest recipe [4]. For example, an image of cake will be further specified as either chocolate cake

or strawberry cake based on the distance between the encoding of the input image and the encodings of chocolate and strawberry cakes.

While the above research was conducted on the Recipe 1M+ database, there have also been many food classification projects based on the Food-101 dataset. Some of these can be found in the Appendix (section 8.2 item 1).

3 Data

3.1 Train and validation data

We used the Food-101 dataset for our research. This is a balanced dataset with 101 classes. Each class represents a different food item and has 1000 images. This brings the total to 101,000 images in the dataset. This dataset also has some noise in the form of intense colors and mislabelled images [5]. Of the 1000 images in each class, 750 are used for training data and 250 are used for validation data resulting in a 75% train and 25% validation split. Figure 1

Figure 1: Images and their corresponding class labels from Food-101

3.2 Test data

3.2.1 Webscraping

We developed a custom web scraper using Google's Programmable Search Engine. We leveraged the Programmable Search JSON API to systematically extract images from the internet based on tailored search criteria. This process included creating a personalized search engine, specifying relevant websites, and configuring parameters to filter image results. Notably, we restricted image collection to those captured post-2014.

To ensure ethical and respectful data acquisition, we implemented measures to skip image URLs associated with a status code 403 (Forbidden). The data-cleaning process involved eliminating duplicates and manually verifying the correctness of the images for each label. Despite ethical constraints on image acquisition, our dataset comprised over 700 images spanning the 101 food classes.

3.2.2 Data leakage

Since we are collecting our test set using a web scraper, we needed to take steps to prevent data leak between the train/validation set and the test set. The Food-101 dataset was created in 2014, so our initial solution was to ensure that the images collected for the test set were past this year. However we later realized that there is still a possibility of some leakage since images from within the Food-101 dataset could be re-posted at later dates. While we were not able to address this issue due to time and computational constraints, we have brainstormed possible solutions that ensure no leakage occurs between these sets.

Given an image, it is possible to determine the PPI (pixels per inch). We can calculate the largest image PPI in Food-101 and scrape images that have higher PPIs for our test data.

$$\text{PPI} = \frac{\text{Image Width in Pixels}}{\text{Image Width in Inches}}$$

While this process will likely eliminate any data leakage, it is still not foolproof. PPI of images can be altered using techniques such as re-sampling.

There are only two completely foolproof methods of ensuring that there is no data leakage. The first is to allocate part of the Food-101 dataset as the test set and not include those images in the training and validation process. The second is to compare the pixel values of each image that is collected with the pixel values of images in its corresponding class. This way we can ensure that there are no duplicates

4 Model

4.1 Base model - MobileNetV2

We employed transfer learning by utilizing the MobileNetV2 architecture, which had been pre-trained on the ImageNet dataset. The Imagenet dataset is a large dataset of 14 million images that belong to 1000 classes. This dataset was created for use in object recognition research [6].

There are many reason as to why we chose MobilenetV2 to be our base model. While models such as ResNet50 and ResNet101 outperform MobileNetV2 in terms of accuracy, they are much larger models. This means that the time to train and the time to predict are much higher than MobileNetV2. Since the ultimate goal of this project is to deploy a working model on mobile, it is paramount that the model used is fast enough, and compact enough to run in a mobile environment. ResNet50 has approximately 25.6 million parameters while MobileNetV2 has around 3.4 million. This is a stark difference that greatly improves the speed of MobileNetV2.

The drastic decrease in parameters is due to the introduction of depthwise separable convolutions. This involves applying a depthwise convolution followed by a 1x1 pointwise convolution. The result is a feature map that captures more intricate patterns within the input data with less layers and parameters. Figure 2 shows the basic structure of MobileNetV2

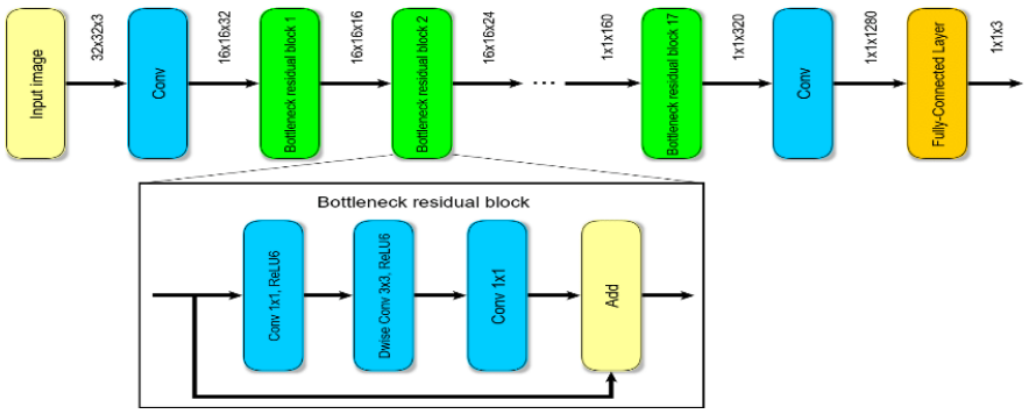

Figure 2: Basic architecture of MobileNetV2

4.2 Model head

When doing transfer learning with CNNs such as MobileNetV2, the base of the model is the convolutional layers and the head of the model is the fully connected layers at the end. The weights of the base model are frozen while the head of the model is restructured and retrained to better fit the specific task at hand. Figure 3 shows the head that we constructed for our model.

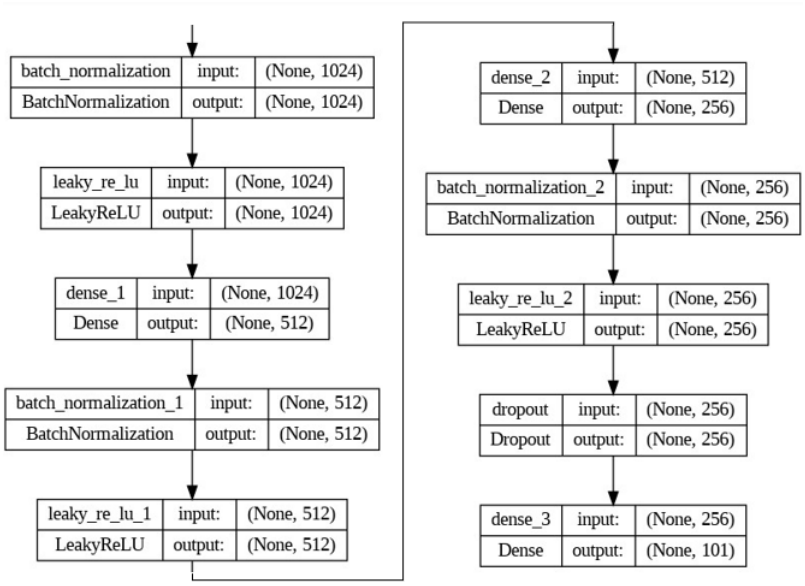

Figure 3: Head of the transfer learning model

4.2.1 Improving generalization

We used batch normalization, regularization, dropout, and learning rate scheduling to improve the generalization of our model. Batch normalization was added to each layer in the head to normalize the input to have a mean of zero and standard deviation of one.

We also chose LeakyReLU as our activation function between layers in the head instead of ReLU which is used throughout the MobilNetV2 convolutional layers. This was because we found that ReLU was causing our model to underfit. LeakyReLU preserves the negative weights that occur while ReLU makes these weights zero. When the weights are zero, they no longer contribute to the prediction and thus certain features are lost.

L2 regularization was added to each layer as well. Similar to ReLU, L1 regularization was causing weights to go to zero. This was hurting the generalization of our model. So we chose L2 regularization which compresses the weights down to smaller numbers but does not make them zero. This preserves learned features.

Finally the amount of dropout used was a big factor. Too much dropout and the model was not learning enough from the train data and thus underfitting. Meanwhile, little, or no dropout was giving us very low validation accuracies.

5 Metric

The function that we used to calculate loss is sparse categorical crossentropy.

$$L(y, \hat{y}) = -\sum_{i} \mathrm{I}(y = i) \cdot \log(\hat{y}_i)$$

The difference between categorical and sparse categorical cross entropy is the format of the input. In categorical crossentropy the input is expected to be one-hot encoded vectors while in sparse categorical crossentropy, the input is an integer representing the class. For this reason the formula includes an indicator function I(y=i). This indicator function returns 1 when the label y equals i and 0 otherwise. i here represents the integer label of each class and $\hat{y}$ represents the predicted probability for class i. The in-sample error (E_{in}) would be the average loss over all training examples (N).

$$E_{\text{in}} = \frac{1}{N} \sum_{j=1}^{N} \left(-\sum_{i} \mathrm{I}(y^{(j)} = i) \cdot \log(\hat{y}_i^{(j)}) \right)$$

The learning rule can then be represented by taking the gradient of this in-sample error. η represents the learning rate.

$$w \leftarrow w - \eta \frac{1}{N} \sum_{j=1}^{N} \nabla_w \left(-\sum_{i} \mathrm{I}(y^{(j)} = i) \cdot \log(\hat{y}_i^{(j)}) \right))$$

Figure 4 shows the accuracy and loss curves using sparse categorical cross entropy. We also ran experiments with KL divergence and sparse categorical

hinge loss. The results were strictly worse than sparse categorical crossentropy. The best model we constructed had a 61% validation accuracy and 60% test accuracy. The loss and accuracy graphs are shown in the Appendix (section 8.2 item 2). We allowed some overfitting to occur because we no-

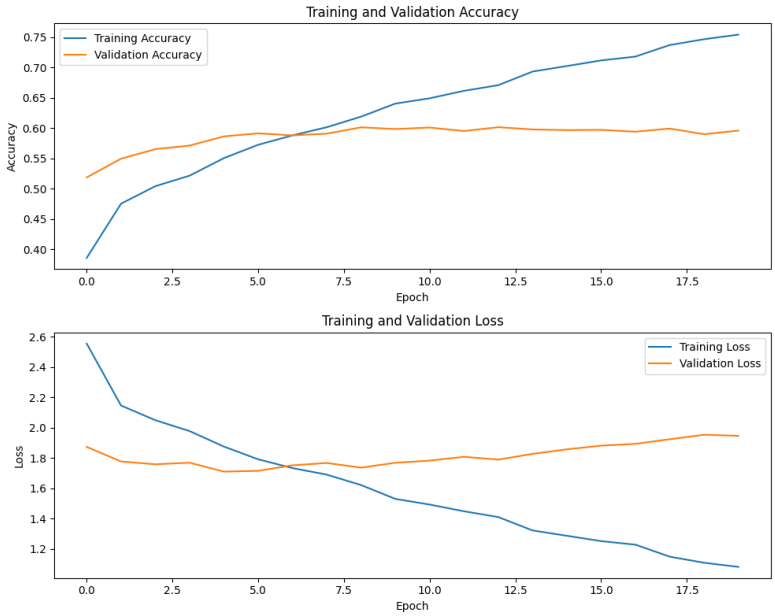

Figure 4: Best results using Sparse Categorical Crossentropy

ticed that while the val accuracy stabilized at 60%, the predictions were much closer to being correct. For example, a model that was not overfitting could have classified french fires as pizza, but the overfitted classifies the french fries as poutine which is a mistake that could be made by humans as well.

6 Mobile Application Development

6.1 Model Conversion

Our model was built using Keras. Keras models cannot be directly deployed on a mobile application. This limitation is due to the model's substantial file size, which is impractical for mobile devices with limited storage capacity.

To overcome these challenges, we first convert the Keras model to TensorFlow Lite (TFLite) format. TensorFlow Lite is designed for on-device

inference and is compatible with a variety of mobile devices. The TensorFlow Lite model format also allows us to leverage on-device accelerators like GPUs if they exist to further improve our model's performance.

6.2 Model Quantization

Quantization is a crucial technique to improve our model's ability to process images. This process involved reducing the precision of the model's weights and activations, resulting in a more compact model without significant loss of accuracy. Our model head parameters are represented as 32-bit floating-point numbers, which consume significant memory. Quantization involves reducing the 32-bit floating-point numbers to lower bit-width representations of 16-bit fixed-point numbers. This reduction precision means fewer bits are involved in computations. This lower computational complexity leads to faster execution of neural network layers during inference, improving overall processing speed.

6.3 Model Deployment

The quantized model was integrated into an Android application developed using Android Studio. The application provides users with the ability to capture images or select them from the device's gallery, and the model then processes these images to identify the food and list its ingredients.

7 Next Steps

We would also like to work with the Recipe1M+ dataset as this is a much larger, and noise-free dataset compared to Food-101. This dataset would even allow us to use the MobileNetV3 architecture which is faster and more accurate than MobileNetV2.

Currently the model prints the final ingredient list using a keyword search pulling from a small database of generic recipes. We would like to make this process more robust. As previous research has shown we would like to build upon the KNN methodology of constructing a recipe list given images. Developing the recipe generation aspect would be the main focus of the next phase of this project.

8 Appendix

8.1 definitions

- **Multimodal Learning** - Models that can work with and relate data that is in different modes (eg. images and text).
- **Joint Neural Embedding** - Data from different modalities (eg. text, images, audio) are embedded into a shared representation space.
- **Representation space** - The mathematical space where data is transformed or mapped to during the learning process
- **Bi-directional LSTM** - Traditionally LSTMs only process sequences from beginning to end, but bi-directional LSTMs also consider data from the end to the beginning. This helps them capture dependencies and patterns in both directions.
- **Images' Encoding** - Image represented in a numerical way that can be used by a machine learning model
- **Balanced Dataset** - A dataset that has the same number of data points in each class
- **Re-sampling** - The process of changing the number of pixels in an image. This can involve increasing or decreasing the resolution of the image. There are two main types of resampling: upsampling (increasing resolution) and downsampling (decreasing resolution).There are many methods for both.
- **Feature map** - The output after applying convolution operations to the input such as multiplying with a filter.
- **Depthwise convolution** - Where each input channel is convolved with its own set of filters independently as opposed to traditional convolution where each filter spans the full depth of the input. This means that for each channel in the input, there is a separate 2D convolutional filter. This operation does not mix information across channels.
- **Pointwise convolution** - The pointwise convolution is just a 1x1 convolution filter that combines these three channels by computing the linear combination of them and outputs one channel.

- **KL Divergence** Kullback-Leibler Divergence, also known as relative entropy, is a measure of how one probability distribution diverges from a second, expected probability distribution. (formula given in section 8.2)
- **Sparse Categorical Hinge Loss** - Extension of the hinge loss, designed for scenarios where the classes are mutually exclusive. Common for SVMs (formula given in section 8.2)

8.2 Other Items

8.2.1 ITEM 1

This is a kaggle code notebook that compares results of VGG-16, ResNet50, and MobilNetV2 on the Food101 dataset
This is a research paper where the researchers used the Food-101 dataset and MobileNet architecture to classify Indian food

8.2.2 ITEM 2

The sparse categorical hinge loss for a single example:

$$L(y, \hat{y}) = \sum_i \max(0, 1-(\mathrm{I}(y = i)-\hat{y}_i))$$

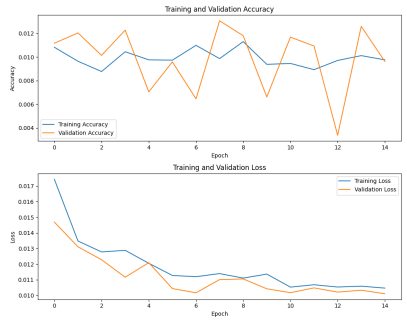

Figure 5: Results of Sparse Categorical Hinge Loss

The KL divergence between two probability distributions P and Q:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

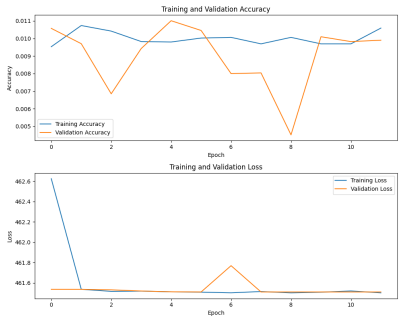

Figure 6: Results of KL Divergence

9 Citations

[1] "Travel and Tourism Market." Transparency Market Research. Accessed December 6, 2023. https://www.transparencymarketresearch.com

[2]MIT - Marin, Javier, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. "Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images." IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. Accessed December 6, 2023. `http://pic2recipe.csail.mit.edu/tpami19.pdf`

[3]1jinwoo. "1jinwoo/Edible: Columbia Devfest 2018 Hack *Best Health Hack Winner*." GitHub. Accessed December 6, 2023. `https://github.com/1jinwoo/Edible`

[4]Recipenet: Image to recipe/nutritional information generator. Accessed December 6, 2023. `http://cs230.stanford.edu/projects_winter_2020/reports/32552727.pdf`

[5]community, The HF Datasets. "FOOD101 · Datasets at Hugging Face." food101 Datasets at Hugging Face. Accessed December 6, 2023. `https://huggingface.co/datasets/food101`

[6] "Wolfram Neural Net Repository." MobileNet V2 - Wolfram Neural Net Repository. Accessed December 6, 2023.
https://resources.wolframcloud.com/NeuralNetRepository/