NutriSnap

CMPT 353 Project Report

Group Members

Sid Agrawal (301207053)

Joshil Patel (301305784)

Sonal Unadkat (301110334)

Problem

The goal of this project is to utilize transfer learning and retrain Google's Inception V3 model to accurately classify images of food and produce the corresponding nutritional information. Using TensorFlow, we were able to obtain a retrain.py script that retrains the final SoftMax layer to classify images specific to this project. The idea behind this project is to use Retrain.py and modify it by adding multiple layers before the final soft-max layer, in hopes of increasing accuracy and stability of the model.

Data: Gathering & Cleaning

We obtained our data using Kaggle and a Google Chrome plugin called FatKun. These sources provided us with our database of 100,000 photos of different types of food, sorted in their corresponding classifications (101 classes, 1000 images/ class). We created a python script move_images.py that will automatically sort through our images, dividing it into test and train directories. This script picked out 75% photos of each food class for our training set, and the remaining 25% for our testing set. This training set was only used when created our proof-of-concept Keras model. When we started using transfer learning using TensorFlow's retrain.py, the program was already configured to create its own testing and training data sets.

Because our data was already pre-sorted in its corresponding food class, there wasn't much data cleaning required. However, we did notice that there were many outlier images in our database. For example, in the Pad Thai folder, we found picture of chicken nuggets. Therefore, as part of data cleaning, we had to go through each folder in our directory of images, and manually delete pictures that didn't make sense.

Once our image data was ready, transformations and distortions were applied to our images in TensorFlow's Retrain.py script. Some of the transformations performed are flipping, scaling, cropping, and modifying brightness. Using this sort of data augmentation is a great way of increasing the training data and variation. In a real-world scenario, not all pictures are taken perfectly, for example some pictures are blurry and/or have different angles. The benefit of performing these transformations is that it can help emulate a real-world scenario by adding variations of our data set and provide a more natural learning set for our mode.

Environment & Configuration

The GPU enabled version of TensorFlow was installed in its own Anaconda environment. GPU's concurrently performs multiple numerical processing tasks which is useful for machine learning algorithms, thus enabling us to run retraining multiple times quickly [1].

Our configuration was setup on a Windows 10 desktop, with an NVIDIA GTX 780 with 3072MB GDDR5 memory, more than enough to hand our scope for image classification. The GPU enabled TensorFlow takes advantage of CUDA technology, NVIDIA's parallel computing platform and programming model that offers a dramatic increase in computing performance [2]. CUDA Toolkit 9.0 was installed with multiple Environment Variables set up to enable the acceleration.

To prepare the retrain files for GPU support, the TensorFlow session config flag is set to allow_*soft_placement=True.* This parameter is important because it prevents device kernel and variable specification errors from occurring. The *allow_soft_placement* allows the GPU to place variables, thus preventing such errors [3]. An additional parameter *tf.device('/gpu:0')* is also added to set the GPU as the device to be used for the session. Changing 0 to 1 or more is for GPU SLI configurations, where users can utilize more than one GPU at a time.

The overall code:

```
with tf.Session(graph=graph, config=tf.ConfigProto(allow_soft_placement=True)) as sess, \
    tf.device('/gpu:0'):
```

There are additional options to limit memory usage, though adjustments were unnecessary due to the relative smaller scope of this project.

Once all essential tools were installed and configured, the GPU enabled Anaconda environment was activated in which the retrain.py script was executed.

Techniques & Manipulation

One of the goals of this project, is to utilize transfer learning and retrain Google's Inception V3 neural network. To accomplish this, we used TensorFlow retrain.py and label_image.py scripts. The retrain.py script allowed us to train the final SoftMax layer to our specific food classes, while label_image.py allows us to feed our model an image and test its results. We later imported the logic of label_image.py to be able to not only test the model but to also work with our Nutritionix Client to obtain nutritional information.

Retrain.py

Training image recognition models requires large computing power. Instead of training from scratch, transfer learning is used by taking a related existing trained model and reusing it for a new model. This method drastically reduces the amount of required computing power, dropping processing time to only thirty minutes on a person workstation [4].

For food image classification, features would have to be created to do analysis for items such as edge detection, or color histograms, etc. The problem comes from a wide variance in image features. This is where Convolutional Neural Networks (CNN) are used, which are generalized to account for variance. CNN's require large computing power, but Google provides a pre-trained CNN model called Inception, which was pre-trained model using one thousand classes [5]. Access to this model is provided through Google's retrain.py.

In the case of food classes, the process of Transfer learning is used to apply the learning from a previous trained session to a new training session [4]. In the Inception model, images are fed as input to different layers, until an output produces a label and classification percentage. These layers are different sets of abstractions mentioned above, such as edge and shape detection at the beginning, to more abstract layers as the layer get deeper [6]. Food images will be retrained on the final layer to add to the overall knowledge held by the Inception model.

When retraining food image using Inception V3, a bottleneck directory is created to cache the output of the lower layers on physical disk [4]. Caching helps speed up future iterations of retraining so that no recalculations must be remade at the lower layers. Once the bottlenecks are complete, the actual training and classification begins, which takes about thirty minutes to run. Each step is show at the command line, displaying increasing accuracy scores for each class.

To use the new retrained classifier to detect classes of food, a TensorFlow session is created. This session provides an environment to perform operations on the tensor data that was created. In the session, the SoftMax function Tensor is retrieved from the final layer of the model. The SoftMax function uses the final layer to map input data to probabilities of an expected output. When complete, a retrained model is produced, which can be used to make predictions on food images.

Additional Trials

Seeing the results of the baseline trial, we conducted 4 additional trials with different combinations of layers before the SoftMax layer. All additional layers were added after the bottleneck layer was created, and before the soft-max layer. Based on previous research, it is suggested that soft-max is the ideal activation function for image classifications and therefore we thought it would be best to add other activation function layers before it. Each model contained a combination of 4 additional activation functions: Sigmoid, Relu, Leaky Relu and Tanh. As seen in Figure 1 below, the red box depicts where exactly the additional activation layers are being inserted.

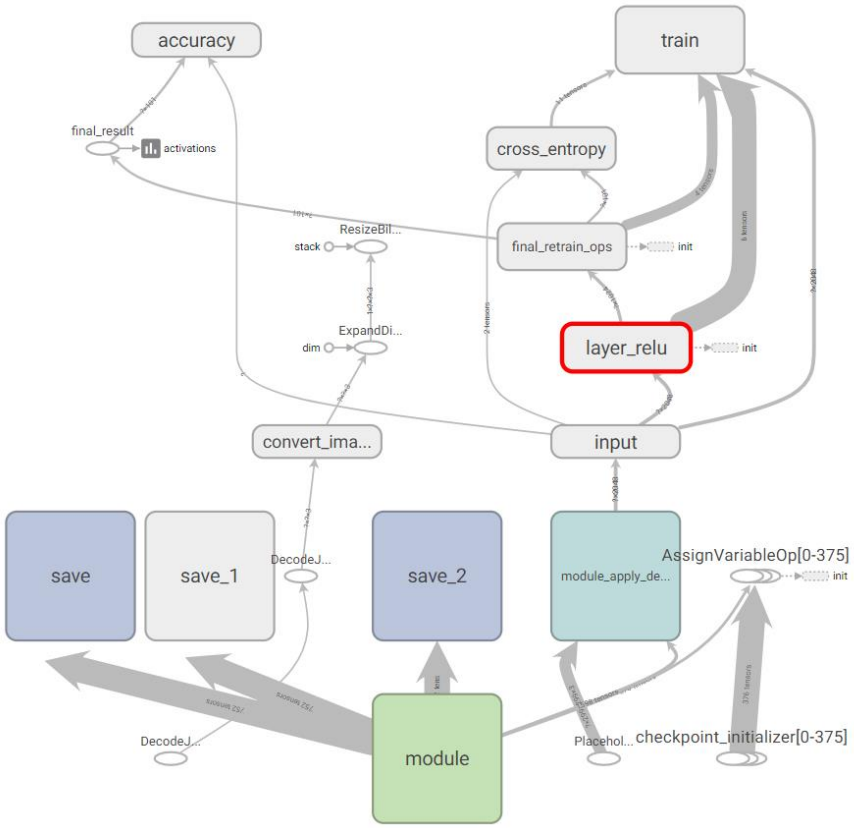

Figure 1: TensorBoard Graph: Visualization of a modified Google Inception V3 Retrained Model.

Trial Overview:

Baseline

Model: Google Inception V3
Additional Layer: NA
Final Layer: SoftMax

Trial 1: Sigmoid & Relu

Model: Google Inception V3
Additional Layer 1: Sigmoid (Bottleneck_size, 1028)
Additional Layer 2: Relu (1024, 1024)
Final Layer: SoftMax

Trial 2: Relu

Model: Google Inception V3
Additional Layer 1: Relu (Bottleneck_size, 1024)
Final Layer: SoftMax

Trial 3: Leaky Relu

Model: Google Inception V3
Additional Layer 1: Leaky Relu (Bottleneck_size, 1024)
Final Layer: SoftMax

Trial 4: Tanh & Relu

Model: Google Inception V3
Additional Layer 1: Tanh (Bottleneck_size, 1028)
Additional Layer 2: Relu (1024, 1024)
Final Layer: SoftMax

Final Trial: (For Android Application Purposes)

Model: Mobile Net (instrumented for quantization)
Additional Layer 1: Relu (Bottleneck_size, 1024)
Final Layer: SoftMax

Testing:

To test our model, each output graph we created was tested with the exact same 7 pictures. For our model we primarily focused on noodle-based pictures because we noticed our model was confusing these classifications in our baseline trial. To standardize each trial, we increased the number of eval_steps, validation percentage, and training percentage in hopes of yielding better and more accurate results. In Figure 2 below, we can see that by increasing the number of eval_steps, we were able to obtain much better accuracy

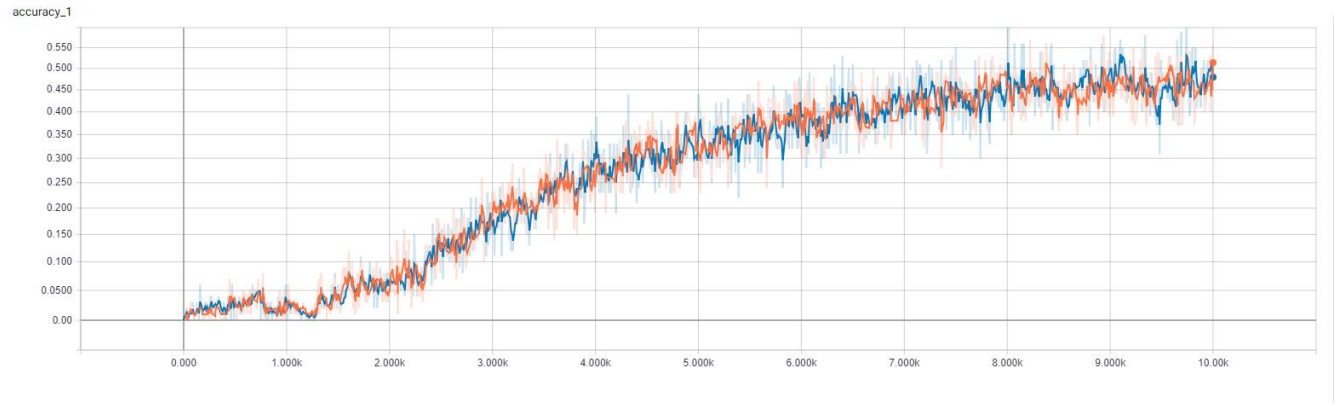

Figure 2: Accuracy Graph depicting increased accuracy with increased eval_steps.

Android Application and Nutritionix Client:

To show the model we created in action as a viable product, we decided to make an android app that would allow us to make predictions of food images. We found a demo Android app made by Google that uses Neural Network models to make image predictions. We downloaded this demo app and used it as the starting point of our app. Modifications were made so that pictures can be taken with the integrated camera, rather than the preloaded images used for the predictions.

Then TOCO, a module within the TensorFlow library, was used to convert our model into a tflite model that could be use in the android app. This step was more complicated than we first thought. We ran into many problems such as figuring out how to convert it, converting it with the correct input and output format, making it small enough for an app, etc.

After much trial and error, we found that the best way to do this was to retrain our model using the Mobilenet pre-trained model which outputted a much smaller graph that works well within mobile apps. After we were able to make predictions using our model within the app, we connected the app to the Nutritionix API. With this we were able to get nutritional info about the food that our model predicted.

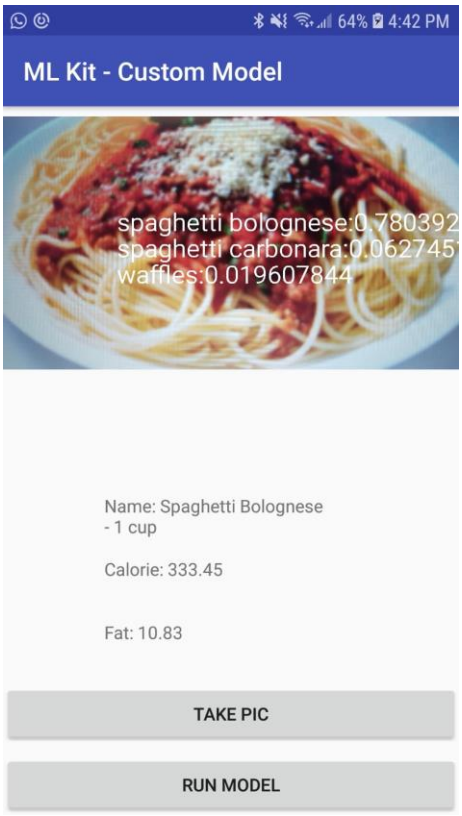

Figure 3: Android Application that can take a picture, classify an image, and display corresponding Nutritional Information.

Results & Conclusions

Trial Results:

After initially running the retrain.py on our dataset, our baseline model was producing results with mixed accuracy. For example, when we asked our model to classify a picture of a burger it produced the correct classification with an accuracy of 0.89. Compare this to a picture of Spaghetti Carbonara, the model classified it as Pad Thai with accuracy of 0.38. We noticed that this model had a lot of issues with noodle -based foods that look very similar, such as ramen and pho.

Trial 1:

For our first trial, we added two layers in our model. The first layer added was a using a Sigmoid Activation Function. This layer took the bottleneck classes and mapped them to 1024 classes. The second layer added right after, was the Relu activation function. The results of this trial were not too promising. For example, hamburger which was obtaining a score of 0.89 in our baseline trial dropped to 0.18. Almost majority of all accuracy scores dropped as well as additional misclassifications. After doing much research, we stumbled across some sources that advised not to use the Sigmoid/Relu combination stating that "using Relu after sigmoid in the last layers worsens the performance of the model." [7] The reason for this could potentially be due to

the affect that this combination has on back propagation. We will also see similar behavior in Trial # 4 when we attempt to use the Tanh/Relu combination in our model.

Trial 2:

For our second trial, we removed the sigmoid layer and kept the Relu activation layer. After retraining our model, we noticed significant better results than the last trial. The model was more accurate in classifying burgers, but also had significantly better luck in correctly classifying our noodle-based images such as pho and ramen.

Trial 3:

For our third trial, we decided to see test if our model was susceptible to "dead nodes". Looking at the Relu activation function, everything that has a negative value will be mapped to 0. Therefore, we thought that it would be beneficial to use leaky Relu that allows for a negative slope rather than mapping to zero., as depicted in Figure 4 below. Comparing our results to the Relu model, we noticed that results were quite similar however, Relu did a far little better at classification than the Leaky Relu. This indicated to us that the dead nodes were not affecting our "Relu" model.

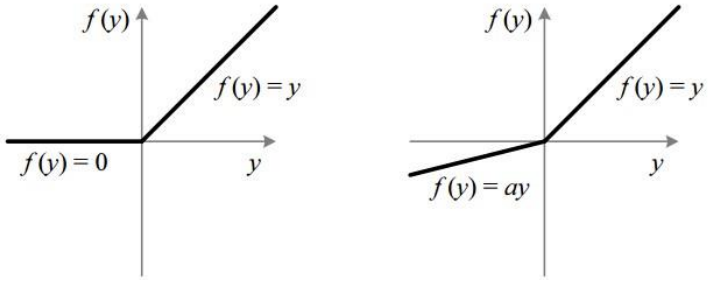

Figure 3: Relu vs. Leaky Relu [8]

Trial 4:

For our last trial, we tried a combination of Tanh and Relu before the SoftMax layer. This trial was not successful at all as the model was classifying everything as "Spring Rolls" with an accuracy of 0.01. We believe that this may be due to the accumulation of dead nodes, as there were only 4 food-classes available for any photo. Our intuition is that Sigmoid and Tanh map values that are less than 0 and combining with a Relu function that maps to 0 is producing many dead nodes in our model.

Photo	(Baseline)	Trial #1 Sigmoid & Relu	Trial #3 Relu	Trial #3 Leaky Relu	Trial #4 Tanh & Relu
Burger	Hamburger 0.89322	Hamburger 0.18	**Hamburger** **0.872**	Hamburger 0.88	Spring Rolls 0.0100
Pho	Bibimbap 0.131	Bibimbap 0.10	**Pho** **0.321**	Pho 0.225	Spring Rolls 0.0100
Ramen	Ramen 0.304	Pho 0.19	**Ramen** **0.47**	Ramen 0.41	Spring Rolls 0.0100
Spaghetti Bolognese	Spaghetti Bolognese 0.29	Pho 0.19	**Spaghetti Bolognese** **0.2461**	Pho 0.23 (Spaghetti Bolognese 0.22)	Spring Rolls 0.0100
Lasagna	Lasagna 0.20	Lasagna 0.09	**Lasagna** **0.31**	Lasagna 0.36	Spring Rolls 0.0100
Red Velvet	Red Velvet 0.65	Red Velvet 0.35	**Red Velvet** **0.41371**	Red Velvet 0.40	Spring Rolls 0.0100
Spaghetti Carbonara	Pad Thai 0.38	Spaghetti Carbonara 0.25	**Spaghetti Carbonara** **0.35**	Spaghetti Bolognese 0.31	Spring Rolls 0.0100

Figure 4: Trial Results. Photo Samples can be found in test pictures folder in our GitHub Repo.

Conclusion

Based on our results, we found the model utilizing the Relu Activation Layer before the SoftMax had the best classification and accuracy. To be able to use with our Android application, we had to re-run the model with Mobile Net model instead of Google Inception V3. This allowed us to easily transform our model into a Tflite model which can be utilized in our application.

Limitations & Additional Problems

Given the limited resources and time that we had for such a complicated problem we ran into many limitations and problems. Firstly, we didn't have enough time, the reason we say this is because training a neural network takes time and if you want to find better neural network layers you must experiment which adds to the time, thus time was a heavy limitation. Next, we were limited in computer resources as the more powerful the graphics card the faster we can train the network and the more steps we can train. Lastly, our neural network and TensorFlow expertise is limited and so there were many things in TensorFlow that we didn't know how to take advantage of that would help our accuracy.

Project Experience Summaries:

Sid Agrawal:

Created a Keras neural network as a proof of concept before diving further into creating a larger neural network.

Found sources for food images that can be used to train the neural network. Transformed the data and put it into the correct format to be consumed by the neural network.

Setup a Tensorflow environment in Anaconda. Took advantage of retrain.py provided by Google to retrain the Inception V3 model to predict 100 different food classes. Trained the Inception V3 multiple times with different layers and activation function to find the one with the best accuracy.

Used TOCO to convert the Graph model that retrain.py outputted into a tflite format that can be used in an Android App.

Created an Android App that can take pictures and uses the tflite model to predict what food is in the picture.

Joshil Patel:

Retrained an Image Classifier for food categories using Googles TensorFlow and Inception V3 architecture. This retrained data was then used to create an Android app called NutriSnap that allows users to take photographs of their meal to predict the class of food and output corresponding nutritional information.

Used CUDA Toolkit 9.0 and modified Google's retrain python script to work with CUDA accelerated graphics on an NVIDIA GPU. This modification drastically reduced computation time needed for training and image classification.

Analyzed and tested four Activation Functions (Relu, Leaky Relu, Sigmoid, and Tanh) to use as layers in transfer learning. Based on the accuracy of the model output, Relu was chosen as it provided the greatest results out of the chosen functions when used to predict food images.

Sonal Unadkat:

Retrained Google Inception V3 neural network using Anaconda and TensorFlow Libraries to be able to classify 101 different food classes. This model was eventually used in an Android application that can take a photo of a food class, classify it, and then display corresponding Nutritional information.

Created a Python Script that can make predictions on a food-image on any given model and match the corresponding Nutritional Information.

Utilized Transfer Learning, to test and analyze 5 different neural networks with different combinations of activation functions as its final layers. Using Relu, Leaky_Relu, Sigmoid, Tanh and SoftMax activation functions we were able to find the most accurate model for our image-classification purposes.

References

[1] TensorFlow, "Install TensorFlow on Windows," 19 July 2018. [Online]. Available: https://www.tensorflow.org/install/install_windows.

[2] NVIDIA, "CUDA Toolkit Documentation," 8 July 2018. [Online]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/.

[3] T. Ganegedara, "Using GPUs Smartly with Tensorflow," [Online]. Available: http://www.thushv.com/tensorflow/using-gpus-smartly-with-tensorflow/.

[4] TensorFlow, "How to Retrain an Image Classifier for New Categories," 2 July 2018. [Online]. Available: https://www.tensorflow.org/hub/tutorials/image_retraining.

[5] TensorFlow, "Image Recognition," 2018 30 July. [Online]. Available: https://www.tensorflow.org/tutorials/images/image_recognition .

[6] Towards Data Science, "Transfer Learning," 23 April 2018. [Online]. Available: https://towardsdatascience.com/transfer-learning-946518f95666.

[7] Towards Data Science, "Is ReLU after Sigmoid bad?," 11 March 2018. [Online]. Available: https://towardsdatascience.com/is-relu-after-sigmoid-bad-661fda45f7a2.

[8] Towards Data Science, "Activation Functions: Neural Networks," 6 September 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.