**APPLICATION OF DEEP LEARNING IN IMAGE CLASSIFICATION**

AN IDEA TO REALITY

‘[**Neural Chef**](https://neural-chefs.herokuapp.com/)’ – Simply put in layman’s terms means ‘render pictures of veggies into recipes using neural networks’. Ever since we embarked on a journey to explore the Artificial Intelligence realm, we realized that the growing sphere of deep learning and computer vision was much more fascinating and exciting. When it came to choosing an intriguing topic for the project, we had to make sure it was insightful, intuitive, amusing and most of all practical. Now, the problem at hand was how we go about accomplishing the objective. Our immense eagerness of putting the theoretical knowledge of deep learning into perspective, directed us towards computer vision.

According to an article by Raj Talluri, “Computer vision is fundamental for a broad set of Internet of Things (IoT) applications. Household monitoring systems use cameras to provide family members with a view of what’s going on at home. Robots and drones use vision processing to map their environment and avoid obstacles in flight. Augmented reality glasses use computer vision to overlay important information on the user’s view, and cars stitch images from multiple cameras mounted in the vehicle to provide drivers with a surround or “bird’s eye” view which helps prevent collisions. The list goes on.”

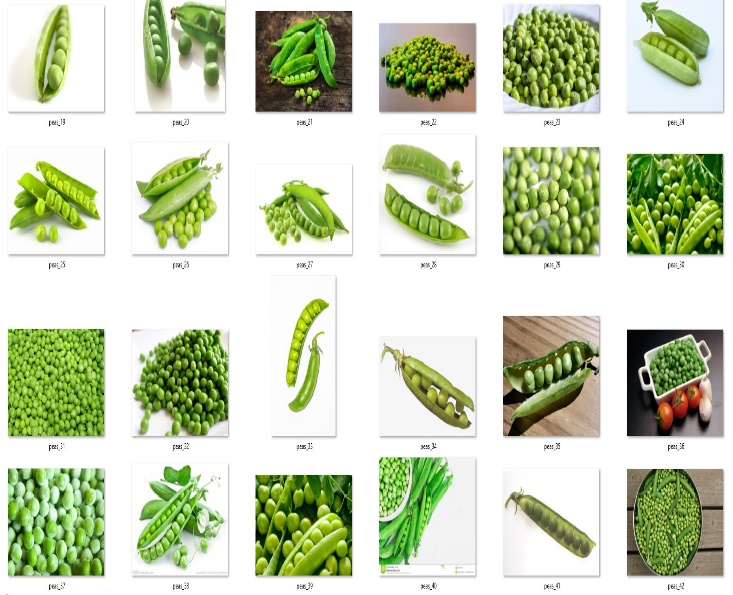
Albeit, virtual web assistants do not fall into the category of computer vision, the concept of gathering the voice commands that follow and sending them to a natural voice recognition service in the cloud, which interprets them and sends back the appropriate response was quite intriguing. Like so, all our minds was racing with a desire to innovate. We, in our team are very much into good food and home cooking, so coming up with some idea along those lines was appetizing. Ultimately it dawned on us: What can be more appealing and tempting than ‘Delectable Recipes’ which we can make at the comfort of our home with the ingredients we have on hands!! For any recipe to be tantalizing, the basic ingredients is ‘Veggies’. What if we created an internet application, wherein the user can upload images of their desired produce and get appropriate healthy recipe recommendations in return ? Voila` , that’s how the app NeuralChef came to being.

To be able to achieve this goal, Convolutional Neural Network (CNN) was our go to. We then decided to take a more refined approach of leveraging a network already pre-trained on a large dataset. Such a network would have already learned features that are useful for most computer vision problems and leveraging such features would allow us to reach a better accuracy than any method that would only rely on the available data. Of the five main state-of-the-art pre-trained networks for image classification, we chose to try out VGG16, VGG19 and the Xception models which have been fully integrated into the Keras core. Although image classification model is heavily-dependent on huge amount of training data, we opted to focus on images of vegetables to keep the memory size optimal for hosting the model online. After all an essential skill of a competent data scientist lies in making the most out of very little data.

DATASET

Following the data-driven approach of accumulating a dataset of labeled images, *produce* images of different categories were scraped from the ginormous [ImageNet](http://www.image-net.org/) dataset and some were also downloaded from Google. Images with only .jpeg extension were sought out. After ascertaining the image quality and sorting, we settled with a target set comprising of 26 different classes (hence no overlap) with each class containing images of varied pixel sizes. The images are clear and have little to no noise (with wee bit of variance) which makes the dataset ideal for this task with considerably much less pre-processing.

A close up of a flower

Description automatically generated

**Sample of produce images from ImageNet and Google**

HARDWARE CHALLENGE

We attempted to train and run the model with our dataset on our CPUs, but the run-time was exasperatingly slow!!! (took days for just 10 epochs). [Google Colaboratory](https://colab.research.google.com/notebooks/welcome.ipynb) made the job easier since it requires no setup and runs entirely in the cloud. TensorFlow on which the Keras runs, speeds up on GPU relative to CPU and much more on TPU and Google Colab gives us that privilege at no charge. The GPUs are very good for matrix-matrix multiplies and have very high bandwidth to memory.

TRAINING AND TEST SET

We used 750 images from all the classes as our training data and 250 images as test data, to evaluate our model. Since the images were hand-picked and manually assigned to their hierarchical folders, the explicit need for splitting the training and test data would seem redundant.

A screenshot of a social media post

Description automatically generated

IMPORTING LIBRARIES

In order to utilize the power of the libraries, we need to import them first.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedTRAINING THE CNN (VGG19 Model)

The VGG19’s architecture includes 5 hidden layers between the input layer and the output. The early layers are convolutional followed by 3 fully connected layers. The activation functions used are rectified linear units (‘relu’) in every hidden layer except for the last layer. The process of training involved:

* Instantiating the convolutional part of the model, everything up to the fully-connected layers and then loading its weights.
* Freeze the first 5 layers of the VGG19 model up to the last convolutional block.

A screenshot of a social media post

Description automatically generated

* In order to prevent overfitting, the last fully connected layer was fine-tuned rather than the entire network. The reason being, the entire network would have a very large entropic capacity leading to a strong tendency to overfit. Overfitting mainly occurs when a model begins to memorize training data rather than learning to generalize from trend. The features learned by low-level convolutional layers are more generic and less abstract than those found higher-up. Hence, it seems logical to keep the first few blocks fixed (with more generic features) and only fine-tune the last one (with more functional features).
* We then run this model on our training and test data, record the output, determine the accuracy for both training and test data and save the model with the parameter save\_weights\_only = True condition.

You can find the entire code for this Model training [here](http://www.google.com).

A close up of text on a white background

Description automatically generated

VGG19 architecture

A screenshot of a social media post

Description automatically generated

Conv block1:

64 o/p filters

Conv block2:

128 o/p filters

Conv block3:

256 o/p filters

Conv block4:

512 o/p filters

Conv block5:

512 o/p filters

Fully-connected classifier

A screenshot of a social media post

Description automatically generatedThe network’s width starts with 64 filters and increases by a factor of 2 subsequently after every Maxpooling layer. Fully connected layer connects the last block of convolutional network to a neural network and ultimately the network gets compiled.

We end the model with 128 units and a different activation function (‘softmax’), which is perfect for a categorical classification. The categorical\_crossentropy loss is used along with it to train our model. In order to keep the previously learned features intact and to make sure that the update magnitude stays very small, fine-tuning is done with a very slow learning rate with the SGD optimizer.

DATA AUGMENTATION

A screenshot of a cell phone

Description automatically generatedAugmentation process was imperative since we did not have a considerably large dataset. In order to prevent overfitting and to help the model generalize better, we ‘augmented’ them via several random transformations. This way our model would never see the exact same picture twice. The ImageDataGenerator class is implemented after compiling the network much before fitting the model to train. This class allows for prepping our dataset by configuring arbitrary transformations and allows for normalization operations to be done on our image data during training. The ImageDataGenerator class also enables to instantiate data generators of the ‘augmented’ batches of images (with their labels) via the .flow\_from\_directory function to be further used for fitting the model to train.

The parameters used in this class are explained very well in the article *Building powerful image classification models using very little data*

* *rescale* is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our model to process, so we target values between 0 and 1 instead by scaling with a 1/255. factor.
* *horizontal\_flip* is for randomly flipping half of the images horizontally
* *fill\_mode* is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.
* *zoom\_range* is for randomly zooming inside pictures
* *width\_shift* and *height\_shift* are ranges (in fraction) within which to randomly translate pictures vertically or horizontally
* *rotation\_range* is a value in degrees (0-180), a range within which to randomly rotate pictures

A screenshot of a cell phone

Description automatically generated

A close up of food

Description automatically generated

**Sample of augmented images after the augmentation approach**

A screenshot of a cell phone

Description automatically generatedThe .flow\_from\_directory() function is used to generate batches of both train and test image data (and their respective labels) directly from our jpgs in their corresponding folders.

These train and test generators are now used to train our model with 30 epochs. Each epoch takes about 300s on GPU but much more on CPU.

A screenshot of a cell phone

Description automatically generated

END RESULT

Owing to the small size of the data set the model turned out to be nearly accurate. We used early stopping to get rid of poor predictive performance (overfitting). We managed to train VGG19 network with about 85% accuracy after 30 epochs (drop out = 0.5). Great success! But with increasing number of epochs, the accuracy will increase.

A screenshot of a cell phone

Description automatically generated

PREDICTION A screenshot of a cell phone

Description automatically generated

And, yes !! our network correctly predicted the image of most of the veggies!! Even though it might not be a 100% accurate invariably, it does render correct predictions in most instances. Adding more convolutional and pooling layers and altering the number of nodes and epochs, might get a higher accuracy result.

EMBEDDING RECIPES API INTO THE APPLICATION

An app was created to implement our successful model and then deployed on Heroku effortlessly. But the main intent of why we started on this endeavor was not yet complete. Hence, our next move was to append the recipe API to our model. The [Edamam API](https://www.edamam.com/) was initially chosen for this purpose but the recipes could not be classified based on the cuisine type. Since we were keen on incorporating a filtering feature for the cuisine type, we used the [Spoonacular API](https://spoonacular.com/) which features tons of healthy and mouth- watering recipes based on several such criteria.

A screenshot of a cell phone

Description automatically generated

**A sample image of the recipe recommender app when given an**

**image of one or more vegetables**

LIMITATIONS OF THE PROJECT

The basic limitations of the project were:

* Data Acquisition phase - access to good quality images in the preferred categories
* Deployment phase – limitation with respect to uploading large sized-image data file (not >100 MB) while deploying the app on Heroku.
* Embedding Recipes API – limitation on the number of request calls and limitation on the number of recipes per request call since we used free applications.

FUTURE REACH

This project can be further extended by embedding our application into refrigerators and integrating with virtual assistants such as Amazon’s Alexa or Google Home. Such a feature would be a must-have trait in any smart homes these days. We have just wet our feet and are ready to dive into and explore an ocean of infinite possibilities.

SUMMARY

Neural network frameworks and models can be re-utilized across a wide variety of use cases as Deep learning delivers superior versatility. With deep learning, a lot of new applications of computer vision techniques have been introduced and most computer vision algorithms tend to be purpose-specific. In this pursuit, we took the data-driven approach and trained a simple Image Recognition and Classifier. The same concept can be applied to a diverse range of objects with a lot of training data and appropriate network.

Project in a nutshell

* Using a pre-trained image classifier with Keras/TensorFlow
* Image augmentation
* Image classification using VGG19 Model
* Serverless uploads with Heroku
* Embed recipes API into application
* Fully trained produce image predicting and

recipe recommender app

**Data Flow Diagram**

INTERNET**+**

A close up of a logo

Description automatically generated

Images

A cucumber on a table

Description automatically generated

Input Image

Pre-trained Deep Convolution Neural Network on A close up of a logo

Description automatically generated

File System Image Database

VGG19 DCNN

Multi-class model based on the pre-trained model

**vgg19\_4\_arch.json**

+

**vgg19\_4\_50.h5**

A screenshot of a cell phone

Description automatically generated

Edamam API

+

Spoonacular API

**PREDICTION**

A screenshot of a cell phone

Description automatically generated

**NEURAL CHEF APP**

REFERENCES

Abdelbaki, Abdul. “Computer Vision Lab SS16 - P-CNN features for Action Recognition.” *ResearchGate*, Oct 2016, https://www.researchgate.net/publication/309392322\_Computer\_Vision\_Lab\_SS16\_-\_P-CNN\_features\_for\_Action\_Recognition

Agarwal, Yash. “Create your first Image Recognition Classifier using CNN, Keras and Tensorflow backend.” *Medium,*08 July 2018, https://medium.com/nybles/create-your-first-image-recognition-classifier-using-cnn-keras-and-tensorflow-backend-6eaab98d14dd

Challet, Francois. “Building powerful image classification models using very little data.” *Blog.keras.io*, 05 June 2016, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Serifovic, Muriz. “Deep Learning Tutorial — Part I: Translate Pictures of Food into Recipes with Deep Learning.” *Towards Data Science*, 09 Sep 2018, https://towardsdatascience.com/this-ai-is-hungry-b2a8655528be

Talluri, Raj. “Conventional computer vision coupled with deep learning makes AI better.” *Network World*, 27 Nov 2017, https://www.networkworld.com/article/3239146/internet-of-things/conventional-computer-vision-coupled-with-deep-learning-makes-ai-better.html