CAB420 - Assignment 2

Group 16

Callum McNeilage - n10482652

Connor Smith - n9991051

Bede Fitzsimmons - n10479422

Queensland University of Technology

<u>Table of Contents</u>

Object Recognition - Caltech 256

Introduction

The purpose of this report is to evaluate the performance of different machine learning methods when used with a large image dataset. We were inspired to choose this topic as throughout the semester we have seen Simon test the limits on his machine with large volumes of data. As most of the assignments have been with smaller datasets, we were intrigued to observe how the methods we learned would perform given a large dataset and see what challenges we would face trying to implement them.

Related Work

There have been several approaches in the Machine Learning area using the Caltech-256 image set (Greg Griffin, 2006) for image recognition and object classification. These approaches include Bag of Features (Li et al., 2016), Nearest-Neighbours (Boiman et al., 2008) and Convolutional Neural Networks (Kagaya & Aizawa, 2015). The objective of this assignment is to test our own approaches to these already known approaches. We will be attempting a Convolutional Neural Network, a clustering algorithm, the accuracy of which will act as a simulated 'Bag of Features' classifier and a Non-Neural Network approach (using K-Nearest Neighbours, SVM and Random Forests).

Data

The dataset used is the Caltech-256 object category dataset. This dataset was chosen as it was suggested on the assignment specification sheet and upon further inspection, seemed to be a reasonable choice for our research question. There are a total of 30637 images belonging to 257 categories. The data was structured as 257 folders each containing the images for that category. The images were resized to 100 x 100 to save on memory and to keep all the images the same size. For the non-neural networks and clustering, the data was transformed into numpy arrays and was normalised. The non-neural network and clustering algorithms were then modified further to create a 2d array. This was done by transforming the data from (number of elements, x, y, channels) to (number of elements, x*y*channels). The non-neural networks were too slow to be trained in a reasonable amount of time, thus they used a fifth of the dataset, with a third of that used for testing. For the neural network, the images were loaded using a data generator to save on memory. Using this method, we discovered a quirk in the dataset which was an empty folder titled 'greg'. This folder kept breaking the generator and was removed. For the generator to work correctly, the dataset needed to be restructured. The dataset was split into 2 folders each containing 257 folders, one for each category. 70% of the images were split into one folder to be used for training and the rest into the other to be used for testing.

Methodology

Neural network:

The neural network we initially used was vgg16. This was chosen as it is well known to be useful for image classification. Attempting to train the model resulted in memory errors due to its size. A data generator was created to reduce the strain on memory but the model took too long to train, taking over an hour per epoch. Reducing the batch size and image size didn't change the timing. Google Collaboratory was used to try and increase the speed but using a data generator bottlenecked the speed, resulting in it taking longer to train. A smaller network was chosen, specifically the second network from week 4 example 2. The model was able to train within a reasonable amount of time but was achieving low accuracy, around 15% for testing. More convolutions needed to be added to capture the details but memory issues prevented this. A ResNet was then chosen as it allowed for a deeper network to be constructed and be trainable within a reasonable amount of time. The model's structure can be seen in the appendix. The model was training fast enough that the images were increased to the size 150 x 150 to try and increase accuracy.

Clustering:

For our clustering approach, we used a KMeans clustering algorithm with the number of clusters set to 257 (the number of object types within the Caltech-256 Image set). Before clustering, we plotted the images based on how they are clustered in the dataset and after clustering we plotted the images based on clustering labels in order to compare an unsupervised model against the pre-set supervised categories. We then compared the unsupervised results against the supervised labels to compare the accuracy of the clustering algorithm.

Non-deep learning:

For the non-deep learning, the methods experimented with were KNNs, random forests and SVMs. When attempting to run these on the full dataset, the load time took multiple days without any result. This being said, for these non-deep learning methods, the data was cut down to a fifth of the dataset. Originally, the intended method was an SVM but due to the large amount of input data, even after the data split, it took a long time for the SVM to run. However, KNNs and random forests were not as computationally heavy and ran much quicker so they were included. Because these methods took less time, the hyper parameters were easier to test and finetune on.

Due to the inadequacy of computational power and large number of classes, when attempting to plot a histogram of the data Python returned an error. It would've been useful to see the dispersion of the data over the classes for better hyperparameter selection. Due to this lack of insight, it was a lot harder to select good hyper parameters for the models.

Evaluation and Discussion

Neural network

Epoch	Training accuracy	Testing accuracy	Time
46	77%	32%	32s
47	77%	29%	33s
48	78%	31%	33s
49	78%	30%	32s
50	79%	31%	33s

Figure 1: Last 5 epochs results from the Resnet

After 50 epochs of training, our model produced 79% accuracy on training and 31% on testing. The model could have been trained for a little longer but was taking over 30s per epoch and would take too long to retrain. Compared to the other networks, it has the best accuracy in both training and testing without overfitting. Overall it took just under half an hour to train, being the quickest to train out of all the models.

Clustering:

As can be seen by the scatter plots below and our comparison algorithm, the KMeans clustering was not very accurate with only an accuracy of 13%. The cluster algorithm also took upwards of 4hrs to run in Google Colaboratory.

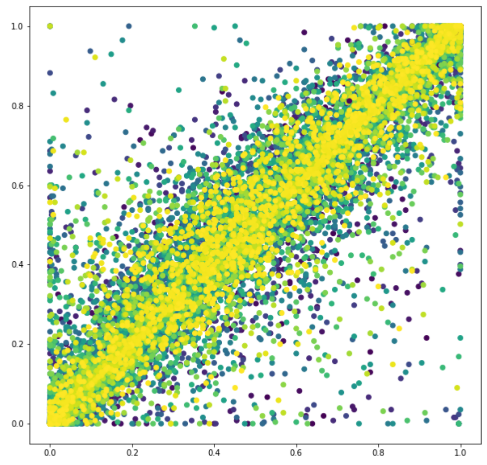


Figure 2: Pre-clustered images of Caltech-256 dataset

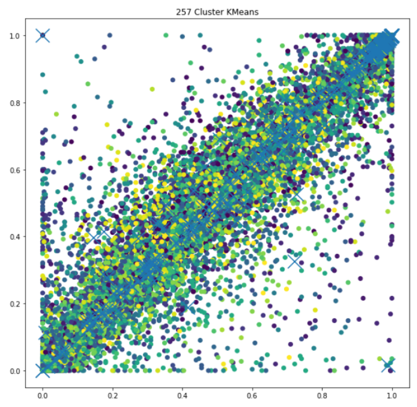
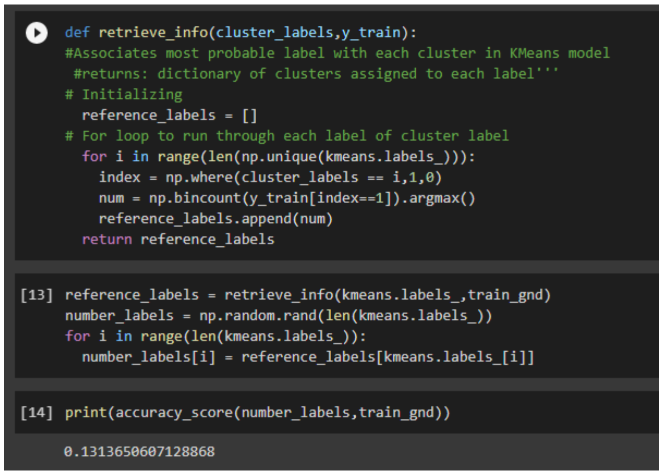


Figure 3: KMeans clustering on Caltech-256 dataset



```
def retrieve_info(cluster_labels,y_train):
#Associates most probable label with each cluster in KMeans model
 #returns: dictionary of clusters assigned to each label'''
# Initializing
  reference_labels = []
# For loop to run through each label of cluster label
  for i in range(len(np.unique(kmeans.labels_))):
    index = np.where(cluster_labels == i,1,0)
    num = np.bincount(y_train[index==1]).argmax()
    reference_labels.append(num)
  return reference_labels
```

```
[13] reference_labels = retrieve_info(kmeans.labels_,train_gnd)
     number_labels = np.random.rand(len(kmeans.labels_))
     for i in range(len(kmeans.labels_)):
       number_labels[i] = reference_labels[kmeans.labels_[i]]
```

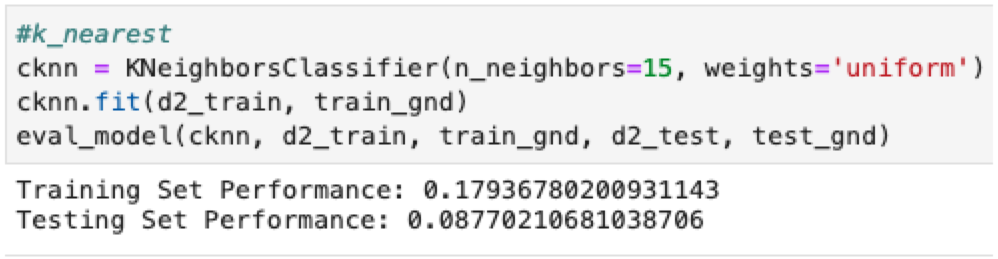
```
[14] print(accuracy_score(number_labels,train_gnd))

     0.1313650607128868
```

Figure 4: Comparison of kmeans.labels_ and Caltech-256 gnd

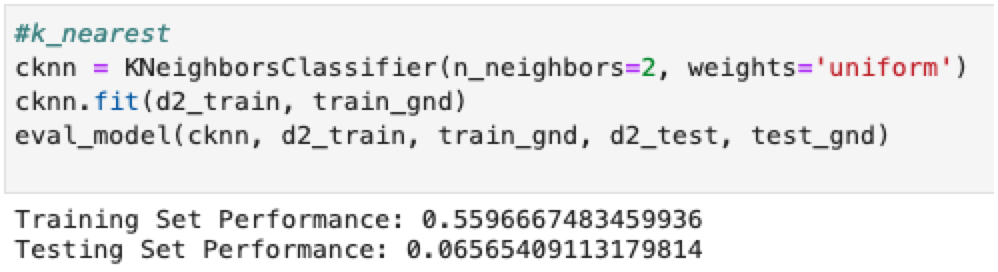
Non-deep learning:

KNN:



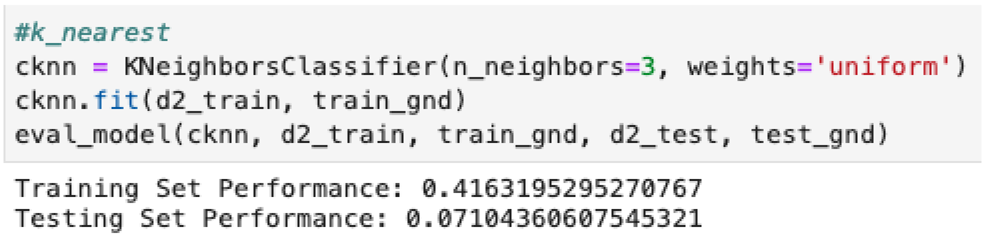
```
#k_nearest
cknn = KNeighborsClassifier(n_neighbors=15, weights='uniform')
cknn.fit(d2_train, train_gnd)
eval_model(cknn, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.17936780200931143
Testing Set Performance: 0.08770210681038706
```



```
#k_nearest
cknn = KNeighborsClassifier(n_neighbors=2, weights='uniform')
cknn.fit(d2_train, train_gnd)
eval_model(cknn, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.5596667483459936
Testing Set Performance: 0.06565409113179814
```

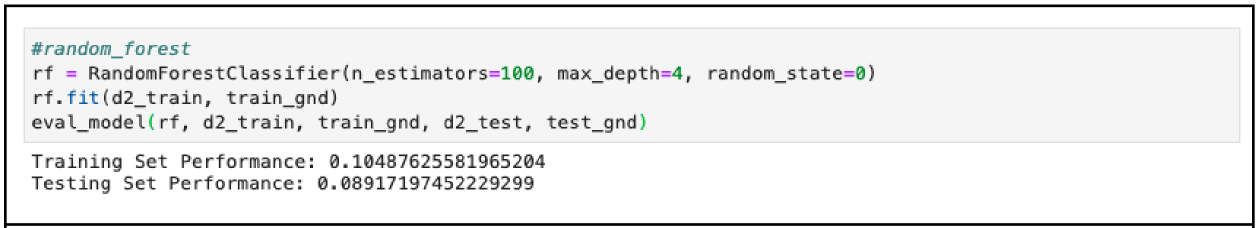


```
#k_nearest
cknn = KNeighborsClassifier(n_neighbors=3, weights='uniform')
cknn.fit(d2_train, train_gnd)
eval_model(cknn, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.4163195295270767
Testing Set Performance: 0.07104360607545321
```

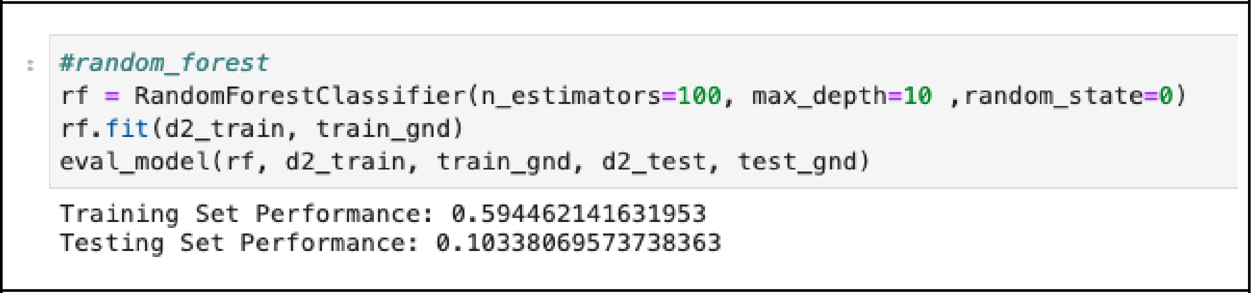
The performance here isn’t great. However, it was hypothesised that the performance wouldn't be stellar since the flattening of a 4D array into a 2D array removed a lot of the meaning behind the images. It was interesting to see that as the number of neighbours decreased, the training performance increased yet the testing performance increased. Although the training performance wasn’t close to maxing out, I doubt that even a high 90% training accuracy wouldn’t be an overfit on the data.

Random Forests:



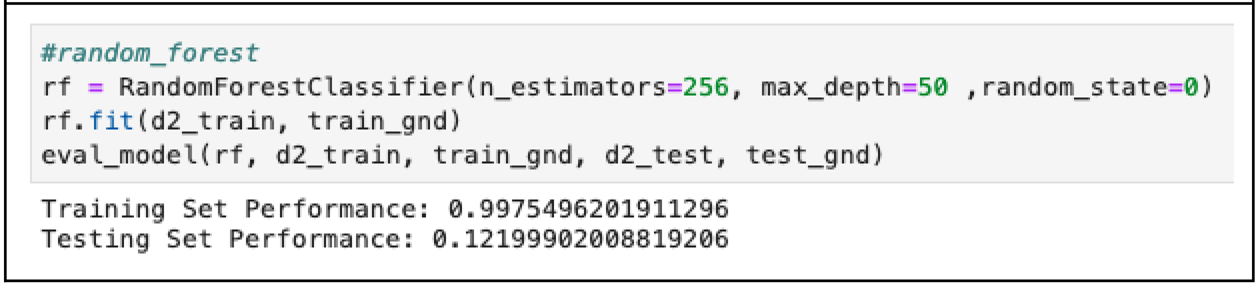
```
#random_forest
rf = RandomForestClassifier(n_estimators=100, max_depth=4, random_state=0)
rf.fit(d2_train, train_gnd)
eval_model(rf, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.10487625581965204
Testing Set Performance: 0.08917197452229299
```



```
#random_forest
rf = RandomForestClassifier(n_estimators=100, max_depth=10 ,random_state=0)
rf.fit(d2_train, train_gnd)
eval_model(rf, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.594462141631953
Testing Set Performance: 0.10338069573738363
```

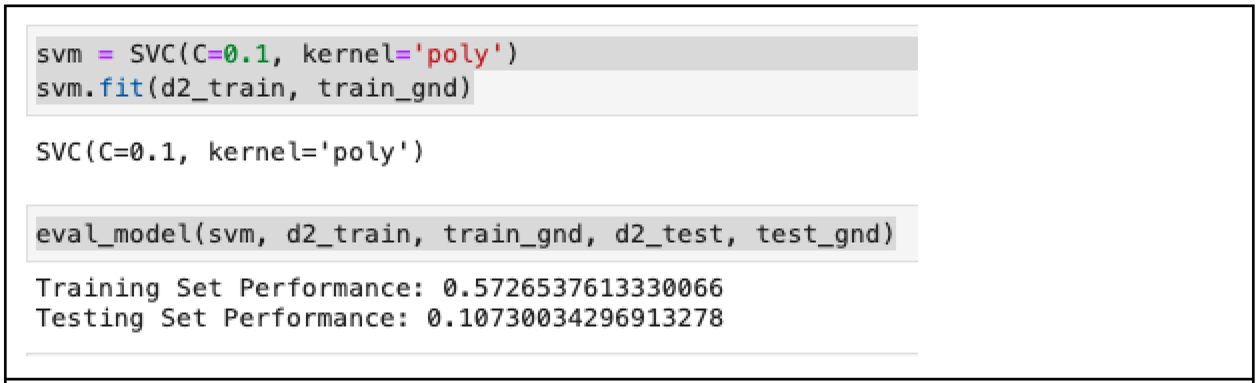


```
#random_forest
rf = RandomForestClassifier(n_estimators=256, max_depth=50 ,random_state=0)
rf.fit(d2_train, train_gnd)
eval_model(rf, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.9975496201911296
Testing Set Performance: 0.12199902008819206
```

The random forest method resulted in slightly better training accuracy but evidently overfit at a low testing performance. It seems that using random forests was a much more suitable option than KNN. That is likely due to the fact that KNNs work better on less dispersed data. Whereas this data has little structure due to the flattening process.

SVM:

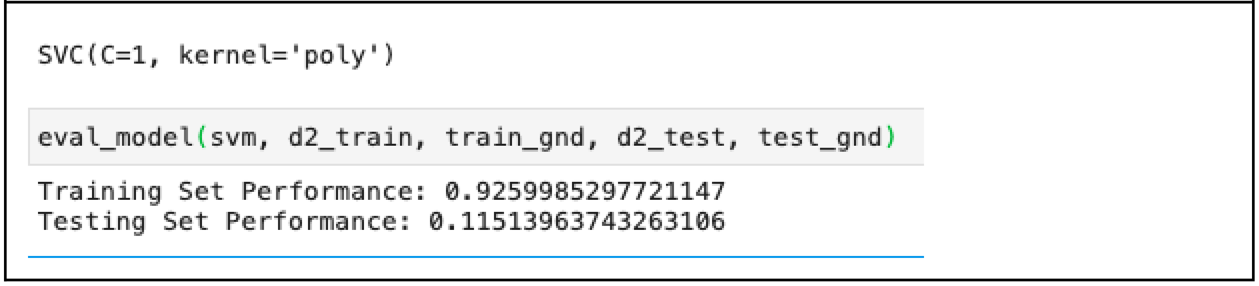


```
svm = SVC(C=0.1, kernel='poly')
svm.fit(d2_train, train_gnd)

SVC(C=0.1, kernel='poly')

eval_model(svm, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.5726537613330066
Testing Set Performance: 0.10730034296913278
```



```
SVC(C=1, kernel='poly')

eval_model(svm, d2_train, train_gnd, d2_test, test_gnd)

Training Set Performance: 0.9259985297721147
Testing Set Performance: 0.11513963743263106
```

The SVM didn’t garner any amazing testing performances and fell behind the random forests’ performance. However, it did eventually fit to the training data well. This normally wouldn’t be anything to celebrate but given that each SVM run took upwards of 15 hours on the machine in use, it was good to see that the hyper parameters were in the right ballpark to fit to the data; albeit an overfit.

Summary:

Overall, the best model in terms of performance and time is the neural network. This makes sense as it is generally known for being useful in image recognition based problems. The ability to use a data generator also helped to reduce time and memory constraints by a lot. Clustering was an interesting choice for a model as it's possible but not first thought of when it comes to image recognition. The performance was quite bad and it was painfully slow, taking 4 hours to train.

There are several reasons we have determined as to why our clustering is not performing very well. Firstly, the number of images of a given object is not uniform for all objects in the dataset with a minimum number of 80 images in a subset. This may have created some issues when we cluster as KMeans clustering does not perform well on non-uniform datasets (Sharma & Yadav, 2016). In addition, images in a given subset can vary drastically in visibility of the object it is supposed to be showing.

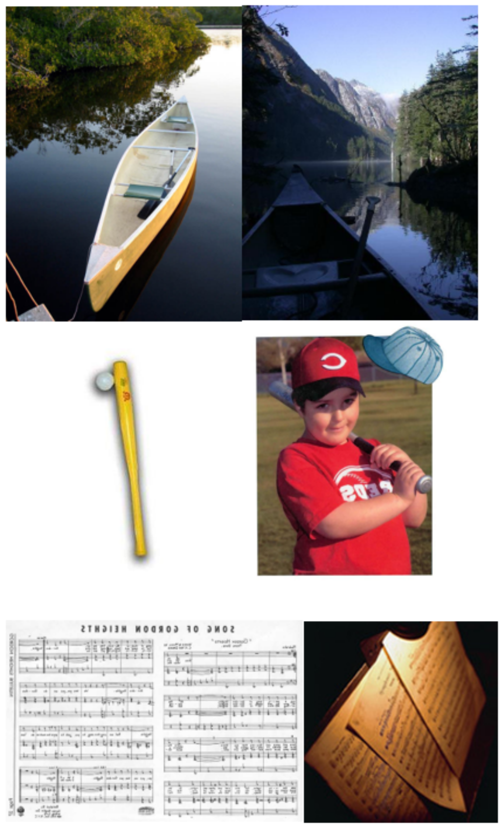


Figure 5: Images within the Canoe, Baseball bat and Sheet Music sets

This will also make it difficult for KMeans to cluster as having such a drastic difference in images will mean that the KMeans algorithm might position the images further away and misclassify them as different categories. These issues can explain why all the models performed poorly. The non-neural networks chosen are known for classification, but not with image datasets. The way the data needed to be structured to be inputted into these models would cause a lot of data to be lost. Both clustering and the non-neural networks also focus on the representation of the whole image rather than the features within them. This explains the better performance from the resnet as its deep structure allowed it to pick up on some of the smaller features.

In regards to the three non neural network methods, each of them struggled to comprehend the data. This is not at fault of the methods but more so a result of flattening a 4D array into 2D. All of the meaning was lost from the original images hence giving the non neural network methods an initial disadvantage. With more hyper parameter tuning there may be an increase in performance but it would never result in the model learning the input data to an acceptable level for image classification

Conclusions

Overall, working with large datasets has proved to be a big issue. The main problem arose from memory issues and timing issues. Neural networks are the reasonable choice when working with large image datasets as it allowed for data generators and worked reasonably quickly. The major shortcoming with our approach was severely underestimating how long models would take to train. With some models taking over 2 hours to train, it made it difficult and draining to experiment with different options. If this approach was to be taken again in future investigations, some manipulation of the data to reduce size may be used. More neural network types would also be tested as they produced the best results for this type of data.

References

Boiman, O., Shechtman, E., & Irani, M. (2008). In defense of Nearest-Neighbor based image classification. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8. https://doi.org/10.1109/CVPR.2008.4587598

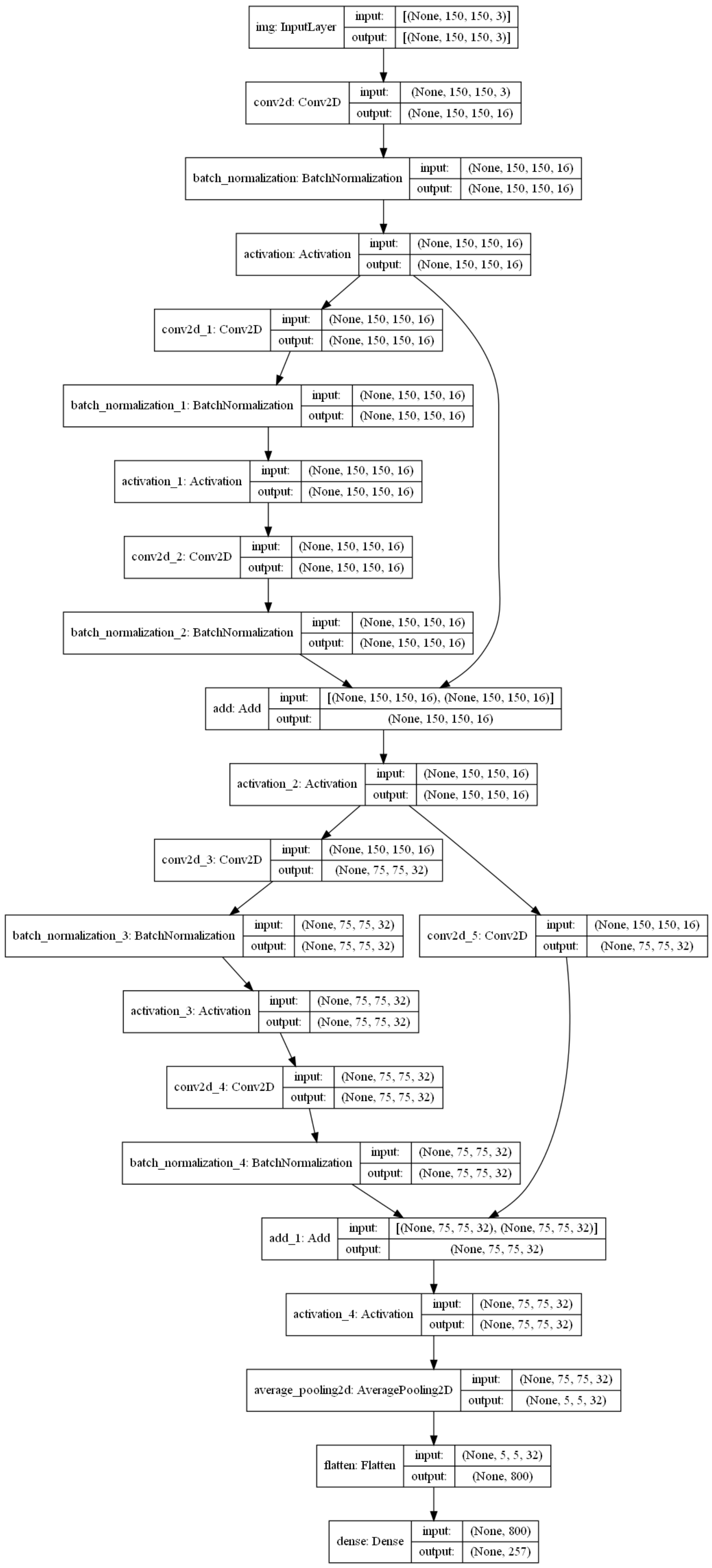
Greg Griffin. (2006, September 27). *Caltech256*. Caltech-256. http://www.vision.caltech.edu/Image_Datasets/Caltech256/

Kagaya, H., & Aizawa, K. (2015). Highly Accurate Food/Non-Food Image Classification Based on a Deep Convolutional Neural Network. In V. Murino, E. Puppo, D. Sona, M. Cristani, & C. Sansone (Eds.), *New Trends in Image Analysis and Processing—ICIAP 2015 Workshops* (pp. 350–357). Springer International Publishing. https://doi.org/10.1007/978-3-319-23222-5_43

Li, K., Wang, F., & Zhang, L. (2016). A new algorithm for image recognition and classification based on improved Bag of Features algorithm. *Optik*, *127*(11), 4736–4740. https://doi.org/10.1016/j.ijleo.2015.08.219

Sharma, G., & Yadav, V. (2016). ANALYSIS OF K- MEANS CLUSTERING ON UNIFORM AND NON- UNIFORM DATA SET. *International Journal of Advances in Electronics and Computer Science*, *3*(12), 4.

Appendix


img: InputLayer | input: [(None, 150, 150, 3)] | output: [(None, 150, 150, 3)]
conv2d: Conv2D | input: (None, 150, 150, 3) | output: (None, 150, 150, 16)
batch_normalization: BatchNormalization | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
activation: Activation | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
conv2d_1: Conv2D | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
batch_normalization_1: BatchNormalization | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
activation_1: Activation | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
conv2d_2: Conv2D | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
batch_normalization_2: BatchNormalization | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
add: Add | input: [(None, 150, 150, 16), (None, 150, 150, 16)] | output: (None, 150, 150, 16)
activation_2: Activation | input: (None, 150, 150, 16) | output: (None, 150, 150, 16)
conv2d_3: Conv2D | input: (None, 150, 150, 16) | output: (None, 75, 75, 32)
batch_normalization_3: BatchNormalization | input: (None, 75, 75, 32) | output: (None, 75, 75, 32)
conv2d_5: Conv2D | input: (None, 150, 150, 16) | output: (None, 75, 75, 32)
activation_3: Activation | input: (None, 75, 75, 32) | output: (None, 75, 75, 32)
conv2d_4: Conv2D | input: (None, 75, 75, 32) | output: (None, 75, 75, 32)
batch_normalization_4: BatchNormalization | input: (None, 75, 75, 32) | output: (None, 75, 75, 32)
add_1: Add | input: [(None, 75, 75, 32), (None, 75, 75, 32)] | output: (None, 75, 75, 32)
activation_4: Activation | input: (None, 75, 75, 32) | output: (None, 75, 75, 32)
average_pooling2d: AveragePooling2D | input: (None, 75, 75, 32) | output: (None, 5, 5, 32)
flatten: Flatten | input: (None, 5, 5, 32) | output: (None, 800)
dense: Dense | input: (None, 800) | output: (None, 257)