REPORT: COURSEWORK 3 - SCENE RECOGNITION

A PREPRINT

Yijia Zhao
MSc Computer Science
yz2g19@soton.ac.uk

Mridula Vijendran
MSc Artificial Intelligence
mv5n19@soton.ac.uk

Run Zhang
MSc Artificial Intelligence
rz2u19@soton.ac.uk

December 13, 2019

1 K-Nearest-Neighbour with *Tiny Image* feature

The experiment was programmed using Python along with the libraries OpenCV[1] and Scikit-learn[2]. *Tiny Image* was used to represent every image concisely by compressing them using bilinear interpolation to a smaller matrix. KNN(K- Nearest Neighbours) was used to classify a test image by comparing it against every training image(that's representative of the true dataset) which eventually returns an answer depending on the class of its neighbouring images via consensus. Apparently, the number of neighbouring K images could make a lot of difference in the model's decision of the class the image in consideration is assigned towards.

1.1 Implementation

We first divided a validation set taken from the training set with ratio of 0.2, on which each class has same amount of images to prevent skewedness of the dataset, thereby affecting what could bias the model towards a particular class.*cv2.reshape*[1] method was used to directly convert all the training images with inconsistent shapes into a set of vectors with same length.If the input image is $N \times N$ then the resize operation will not warp the image contents(change the aspect ratio) and affect the classifier's decision making process. Certainly, same thing would be done to the test and validation set. By default, this method use linear interpolation to reshape the image to target size, thereby allowing faster model processing and allowing the model to work with more important features of the image(since it was compressed). With regard to KNN, we used *KNeighborsClassifier's*[2] fit method with the features array as input and their corresponding labels, which were denoted by class numbers, as output . Besides, we did found that scaling image including zero-mean and unit variance, which could lead to a lower variance in the number of possible outcomes with which the model would have to find the best fit under and with no bias(from the zero-mean), could help improve classifier performance. We achieve this by using the *preprocessing*[2] library to scale the images and use the same parameters for scaling the training set, to scale the test and validation sets. For the *Tiny Image*(preprocessed image) resolution, we used the recommended one 16*16.

1.2 Result and tuning

While dealing with the value of K, we test the accuracy in a scale of values between 1 to 100 as shown in Figure 1. Finally, we pick 5 as the value of K empirically which is likely to be its optimal selection, given that the accuracy peaks at that point.

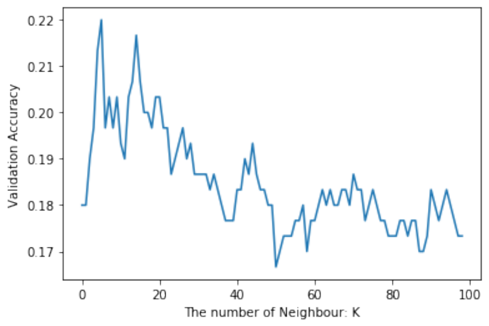


Figure 1: The difference of validation accuracy with different number of neighbors K

2 Linear Classifier with Vector Quantisation of the K-Means Bag-Of-Visual-Word

The whole process's flow diagram is shown in Figure 2. The Training set contains 1200 images out of which 300 has been split into the validation set. The Feature Extraction process follows the recommended fixed size(8*8 patch with a stride of 4 pixels), which is why it resulted into a set of vectors with a length of 64. After standard normalization (zero mean and unit variance) and K-Means clustering of the preprocessed images, there will be K centres, each representing a visual word and made from clustering to common centroids around the input patches from the images, which will be used to do dataset quantisation. A Word Histogram model with a set of vectors representing the occurrence frequency of visual words on images is then passed the quantised data. Since every images have different amount of feature because of different shape as a count, we did standard normalization again, which concludes the preprocessing of the data, and fed to the linear classifier for training. Validation set had also underwent the same preprocessing technique.

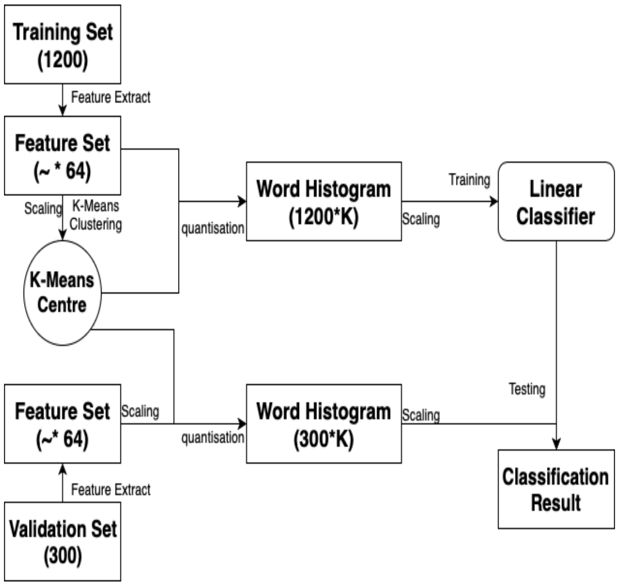


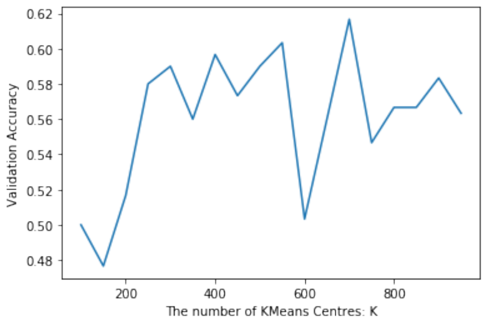
Figure 2: The process flow of run2

2.1 Implement

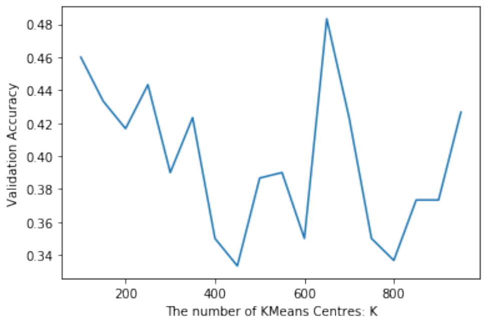
MiniBatchKMeans[2] was used to do clustering since there is a relatively huge amount of data. With respect to the Linear classifier, we used the *OneVsOneClassifier*[2] model after we found the performance of the *OneVsRestClassifier*[2] model was not as good as we expected. With this method, there will be a classifier for each pair of classes, which might required more time to fit and thereby create a better preference ranking of the models for a datum as compared to the latter model. All the classifiers will use the *Linear Support Vector Classification* model [2] to create the maximal margin of separation to reduce the possibility of ambiguous class demarcation and classification.

2.2 Result and tuning

Though the number of centres, the fixed size of the patches, the stride and the different classifier types might affect the final result in varying degrees with certain combinations



(a) OneVsOneClassifier



(b) OneVsRestClassifier

Figure 3: Different validation accuracy with different KMeans Centres

leading to better results than others, due to time constraints in testing out all possible combinations(exponential on trying an exhaustive search for the best model), we only tested the first one's influence on the accuracy. As shown on Figure 3, we eventually set the patch size and the KMeans centre to be 700, which is likely to be the optimal given that the accuracy peaked at that size.

3 Transfer Learning to the classification problem

This experiment was mainly based on the TensorFlow[3] framework. Transfer learning[4] is a way to avoid training a neural network from scratch by using pre-trained weights especially when there is a limited amount of training samples(which is suited for our problem). We have tried different open source fine-tuned models which have been trained on the ImageNet[5] data set which have a lot of images from the real world making these models have a suitable prior upon which we can train to make them more suitable to classify with our dataset. We froze the pre-trained parameters below the fully connected layers of the models and added one dense layer at the end of the network which were associated from the output of the last frozen layer and to the classes we are going to classify (in the form of a dense layer with that many units), which here is 15. Instead of using one models, we thought that different model may extract different features on the images which means their combination might result to a more stable and reliable classifier in the form of consensus of the results which they each provide(taking inspiration from ensemble models). Here, we used the MobileNet2[6] and EfficientNet[7]. The whole structure could be seen at Figure 4.

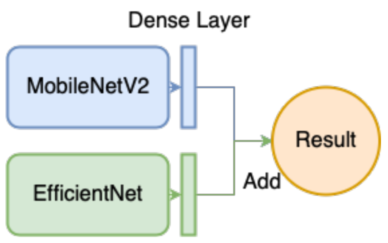


Figure 4: The Model Flow

3.1 Implement

First we used *TensorFlow Hub*[3] to download the latest models with the parameters trained on ImageNet[5]. Then the dense layer was added and trained on the training images. The two model were trained separately. After each model were trained to converge, they are prepared for classification and the result from them will be a vector with length of 15 whose each element denote the probability of the input image belongs to each class. While testing, the predicted vectors from two models will be added up and we got the final predicted result by picking the index corresponding to the max value of the addition of their vectors. In this case, since the convolutional models builds its decision upon the small features and eventually growing in its receptive field across the different layers(increasing the complexity of the features it learns), the model is more robust against the input data having different sizes and aspect ratios.

3.2 Result and tuning

Though we have tried enough amount of different pre-trained models to implement transfer learning in this case, the final validation accuracy always could not reach 95 percent. We believe that this might be a consequence of the model's inability to distinguish between very similar images of different classes(due to most of the details of the images being lost with the receptive field at the later layers of the model look at a wider area, putting less focus on the details of the image).

As it shows on Figure 5, we test the same validation set several times and the accuracy floats around 90 percent. The green line represents the combination of two models' results. Though it shows little improvement and sometimes it was worse than one single model(because the models aren't exactly weak classifiers that compensate for each other's weaknesses), in overall, it is slightly better than a single model's performance. This method is useless when the majority of two model's mis-classification have

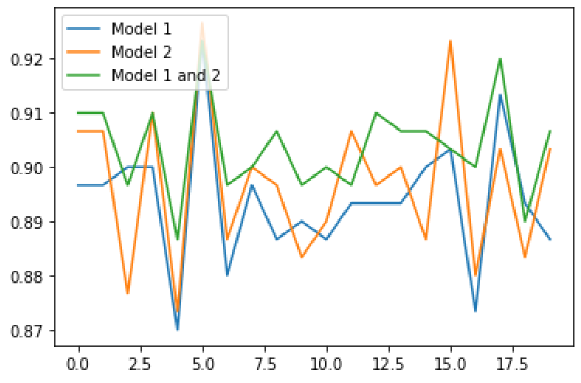
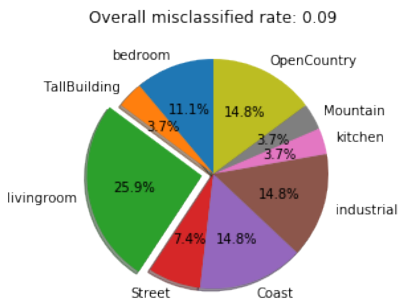


Figure 5: Model 1 is MobileNet and model2 is EfficientNet

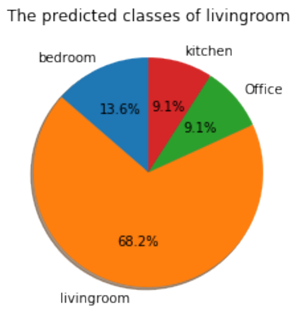
same class(because they retain the first when their probability added up. This can also indicate that the models themselves have an inherent flaw in the way they classify images). However, we found that the correct class probability of those mis-classified samples usually appear in the top3 or top5 on the result vector. And the situation did appear a lot that one model's mis-classified sample could be correctly predicted by another one, which is the reason why their addition could turns out a better classification performance. If there were more models mixed here, the result might have a slight improvement.

3.3 Analysis

When we deal with the classification problem, there was an assumption that the images on each class are basically separable. However, when looking at the training set, we see that even humans could sometimes mis-classify, for example, the mountain on the *OpenCountry* to the *Mountain* class. On Figure 6, it turns out that *living room* is the class our classifier mostly incorrectly predict and three wrong class it usually assign to were *bedroom*, *Kitchen* and *Office* and some of the model predictions to intuitively understand what the model classifies correctly or not is shown in Figure 7. Those classes are intuitively more closed to the *livingroom* than other labels. The separability training classes and the ability of the model to classify objects at different scales in the image(thereby keep the details of the image, while differentiating between noise and detail) are some important factors that may set the limitation of the performance of classification models. Besides, when we need to increase model' performance, we have to find a way to separate the *closed classes*, ie., the classes that are very similar to other or have a majority of its contents overlap with other classes. This could may be reached by training a deep learning model that effectively considers evaluating the model from its different layers(as in focal loss and retina net as in [8]) or we could add second procedure to manually extract more features from those images. There could also be considerations for a Bayesian model that can return the level of uncertainty of classification of a model to allow a more detailed classification using the differentiating or critical features(that could allow disambiguation between overlapping classes) in the image.



(a) Most mis-classified class



(b) which classes were incorrectly classified as living room

Figure 6: The analysis of the result

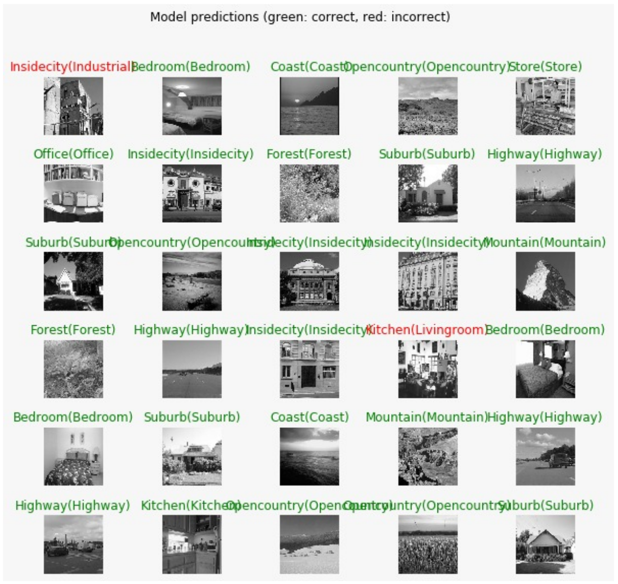


Figure 7: Some examples of correct and misclassified model predictions

4 Individual Contribution

Every member did the first and second runs on their own in order to learn something by it. We set meeting to discuss about run 3 and all attended. Thus, we prefer to say every one has equal contribution.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Martín Abadi and Ashish Agarwal etc. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018.

[7] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

[8] Lin Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 2999–3007, 2017.