COMP6223: Scene Recognition 2019

Brent De Hauwere[1], **Matthew De Vries**[2], **Yanislav Donchev**[3], **Dimitris Mallios**[4], **Pier Paolo Ippolito**[5]
University of Southampton
[1]bdh1g19, [2]mrdv1n19, [3]ydd1g16, [4]dm1n19, [5]ppi1u16
**All authors contributed equally to the writing and research in this study. Their names are listed in alphabetical order.*

1 Introduction

Scene recognition is one of the hallmark tasks of computer vision. The problem of identifying a scene can be approached in a number of ways. Recently, this usually requires the training of deep convolutional neural networks (CNNs) on large datasets. In this assignment, we will experiment with different methods of scene recognition ranging from simple to "state-of-the-art" methods. The dataset we have been given to train on consists of 15 different classes each containing 100 images. This data set is far too small to train a CNN classifier that can generalise. For this reason, we have used different techniques of feature extraction and then classified images based on those features. We have also achieved top accuracy using transfer learning on an ensemble of pre-trained CNN models. For each run, a 70:30 ratio was used for the train test split. The two resulting datasets were ensured to be balanced (have the same number of samples from each class) to prevent biasing of the models. After each model has been tuned, we trained it on the whole training dataset to maximise the number of input representations.

2 Run 1

The "tiny image" feature, inspired by the work of Torralba et al. [25], is one of the simplest possible image representations. We used NumPy's [17] slicing method to crop the original images to a square about the centre, and then OpenCV [1] to resize each image to a smaller representation. We also tried resizing first and then cropping to test if the interpolation from resizing would preserve more information around the edges and improve accuracy. For the resizing part we tested the results for five interpolation methods: nearest-neighbour interpolation, bilinear interpolation, resampling using pixel area relation, bicubic interpolation over 4×4 pixel neighbourhood, and Lanczos interpolation over 8×8 pixel neighbourhood. A range of tiny image sizes from 1 to 32 was tested. We then normalised each tiny image to have zero mean and unit length using scikit-learn [19] and vectorise it using NumPy. The "tiny images" are not a particularly good representation, because they discard all the high frequency image content and are not especially invariant to spatial or brightness shifts. We are using them simply as a baseline. After the processing of the images, a simple k-nearest-neighbour classifier was trained using the scikit-learn [19] library.

2.1 Results

Below is an example output of the methods that perform resizing (by preserving the aspect ratio) and centre-cropping the image to a square.

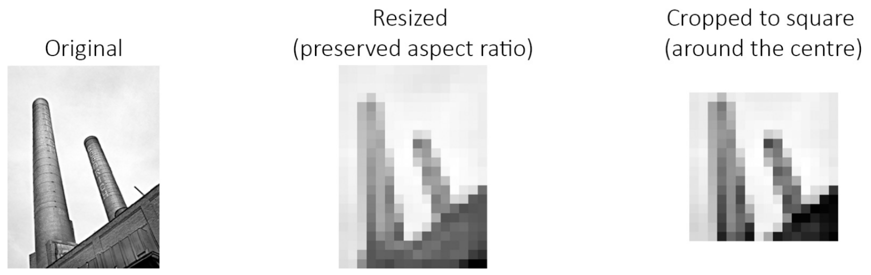

Original
Resized (preserved aspect ratio)
Cropped to square (around the centre)

Figure 1: An example of the image transformations that were applied to create the "tiny images".

Fig. 2 shows the accuracy results of the "tiny image" classifier for a range of square image sizes and interpolation techniques. In this case, resizing was done before cropping. The best accuracy (24.44 %) was achieved with an image size of 11×11 and resampling using pixel area relation interpolation during rescaling. We did the same study for the case where the images are cropped before resizing and achieved a top accuracy of (24 %). It's hard to confirm our hypothesis that resizing first is better since the results are negligible.

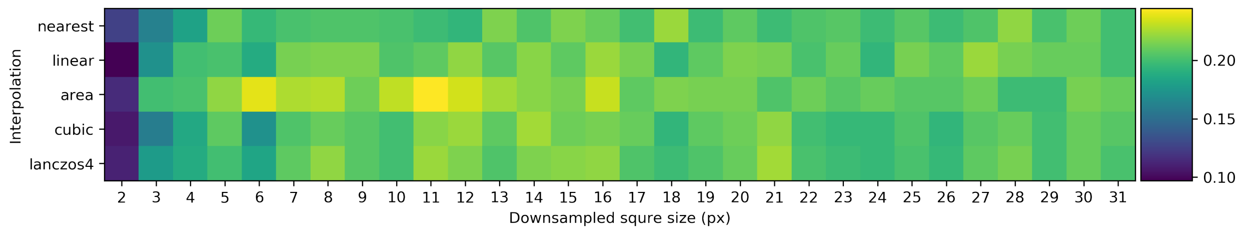


Figure 2: Accuracy map for a range of "tiny image" sizes and interpolation methods.

3 Run 2

A bag of visual words (BoVW) is commonly used in image classification. Its concept is adapted from information retrieval and bag of words used in natural language processing. In computer vision, a BoVW is a vector of occurrence counts of a vocabulary of local image features. There are two steps in generating a BoVW - feature description, and codebook generation.

To build the classifier, a custom method for patch generation was created using NumPy slicing. The method allows for window size and step size adjustment and returns a list of patches. Each patch is scaled to have zero mean and unit length using scikit-learn. To create the vocabulary, the patches from all images are stored in a one-dimensional array and clustered using a mini batch k-means [15] from scikit-learn. We decided to use the mini batch version of the k-means, because it is significantly faster than the standard algorithm but gives similar results [22]. After the k-means model was trained, all patches from each image were quantised by it and the word occurrences were counted in a histogram using NumPy. This process formed histogram features for each image which were used to train a range of classifiers. The optimisation of the window size, the cluster count and the final classifier is described below. The step size was half of the window size in each case.

3.1 Results

We started with the recommended window size of 8×8, a cluster size of 500 and a one-vs-all linear SVM classifier which yielded an accuracy of 51.3% on the validation dataset. A range of different cluster sizes (50 to 750 with a step of 100) was tested (see Appendix A). The best cluster size of 250 increased the accuracy to 53.7%. Different classifiers (see Appendix A) were then trained on the histogram features. The best one (MLP) increased the accuracy to 56.1%. Genetic algorithms from the TPOT library [18] were used to find the optimal solver, activation functions, hidden layer sizes and learning rate of the MLP. Using an Adam optimiser, ReLU activations, 100 hidden units and a learning rate of 0.001 increased the accuracy to 57.2%. Lastly, the window size was optimised by running a sweep of power of two values between 2 and 32. The best window size was found to be 4, which increased the accuracy to 61% on the validation dataset.

4 Run 3

In this section we first introduce our best performing algorithm which was created by transfer learning (TF) on an ensemble of convolutional neural networks (CNNs). We then compare its performance to other algorithms we built – Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT) with BoVW, Dense SIFT in a Gaussian Pyramid, Speeded Up Robust Features (SURF) with BoVW, Fisher Vectors, and lastly applying a set of linear classifiers to the features generated by pretrained CNN models.

4.1 Transfer Learning

We started by removing the fully connected classification layers of ResNet50 (trained on ImageNet [5]) in PyTorch. This exposed the features generated by the network for a given input image. A range of classifiers were trained on these features (see Table 1).

KNN	Linear SVM	RBF SVM	Decision Tree	Random Forest	MLP	XGBOOST	Bagging Classifier
58.66%	87.66%	87.33%	53.66%	40.66%	86.66 %	85%	73.33%

Table 1: Results from a number of different classifiers trained on ResNet50 features.

The top classifiers are the linear and radial basis function versions of SVM, closely followed by a multilayer perceptron (MLP). Although SVM gives a slight advantage in terms of accuracy, using an MLP would allow to fine-tune the CNN and makes ad hoc image augmentation efficient and simple. Therefore, MLPs (fully connected layers) are used from now on.

4.1.1 Image Preparation

All pretrained models in PyTorch accept RGB images of dimensions 224×224 in the range $[0, 1]$. We cropped and resized each image in our dataset accordingly. The single channel of the images was copied three times (to form a three-channel image) and each channel was normalised according the documentation [6]. Furthermore, each time an image was sampled from the training dataset, it was cropped to a random size with a random aspect ratio; then a random rotation and a random horizontal flip were applied. Using this augmentation technique ensured that a repeating image will never be provided to the model, reducing the chance of overfitting [3].

4.1.2 Model Ensemble

To increase the number of features to the MLP, the outputs from 7 pre-trained models were combined. Four of the models – VGG16, ResNet152, DenseNet161 and Inception V3 – were trained on ImageNet. ImageNet has classes such as plants, geological formations and natural objects [5] so we can expect to receive some transferable features. Another dataset, Places365 [28], resembles the one in our problem more closely by providing images (and classes) of forests, fields, cities, etc. The other three models in our ensemble – VGG16, AlexNet and ResNet50 – were trained on this dataset, and their weights were copied on existing PyTorch models. Finally, all convolutional layers of these networks were frozen (non-trainable), all classification layers were randomly reinitialised, and their outputs were connected to a fully connected layer with 15 outputs.

4.1.3 Training and Fine-Tuning

A cross entropy loss function was used to penalise the model. Due to the long training times, optimisation of the batch size was not possible, so we chose a batch size of 16, assuming that it is small enough to escape local optima, but big enough to avoid noisy gradients. For the same reason, learning rate optimisation was impractical. Therefore, a stochastic gradient descent (SGD) optimiser [11] with a learning rate of 0.001 and a momentum of 0.9 was used, but the learning rate would decay by a factor of 0.1 every 5 epochs. As a last measure to over-fitting, we added a condition that would only update the weights of the model if their change improved the accuracy on the validation dataset. Training started by optimising the unfreezed classification layers for 20 epochs. This trained the classifiers problem specific representations without affecting the generic representations from the convolutional layers. After that, the last three convolutional layers from each model in the ensemble were unfreezed and trained for 20 more epochs with a reduced learning rate of 0.0002 and the same schedule for decaying. We did not train the other convolutional layers to preserve the more generic features and prevent over-fitting.

4.1.4 Results

Below are the results from the CNNs ensemble training. The top validation accuracy, achieved after fine-tuning the model is 94.0%. Prior to fine-tuning the maximum accuracy was 93.2%. The gap between the validation and training metrics is small, which means that the model generalised well. In fact, it often achieved worse metrics on the training data because of the augmentation transforms that were applied to it.

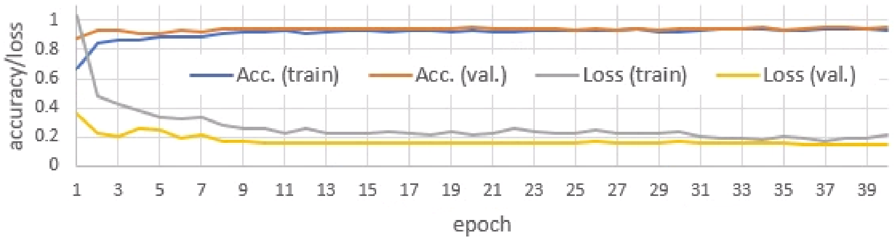


Figure 3: Metrics plot for the training of the CNNs ensemble.

4.2 HOG

HOG is a feature description technique used in computer vision for object detection [4]. What makes HOG a good feature description technique, is its ability to not just detect if a pixel is an edge or not (like in edge detection), but to also provide the gradient and orientation of the edge.

When using HOG feature description, an image is first divided into small portions and for each of these different portions, the gradients and orientations are calculated. Finally, a histogram for each of the different regions is generated.

The HOG features were generated using the Computer Vision Toolbox in MATLAB (using a cell size of 7×7). A one-vs-all linear SVM classifier from the Statistics and Machine Learning Toolbox was fit on these features and achieved 39.6% accuracy on the validation dataset.

4.3 SIFT with BoVW

SIFT descriptors are one of the most fundamental feature extractors for images in the computer vision literature [16]. SIFT are, as the name suggests, invariant to image scale and rotation while being robust in illumination changes and noise. The parameters chosen for the experiments using the SIFT descriptors were set corresponding to the suggested by Lowe [16]. In dense SIFT descriptors are usually computed for each pixel. In our case this was impractical, so we re-scaled and centre-cropped all images. We then took small steps (of 5 or 10) along the x and y axes, keeping the amount of features generated from each image constant. Then a 10% random sample from the training dataset was used to quantise the features and create a codebook using k-means clustering. The size of the codebook is essential for a satisfactory performance of the algorithm. A tiny codebook can be insufficient to represent the image features while a large codebook may not assign similar features to the same cluster. We tried codebook sizes of 500 and 1000. After creating histograms from the SIFT features, linear SVM and random forest classifiers were fitted on them. Table 3 presents the results using BoVW on SIFT descriptors with different classifiers and codebook sizes. The optimal results (59.53% on validation) were achieved using 500 clusters and the linear SVM classifier on features extracted with a step size of 5. In our implementation, we used OpenCV to generate SIFT.

As an extension to the above, we also implemented Dense SIFT in a Gaussian Pyramid using OpenCV and scikit-learn. This feature generation technique was implemented by creating a scale-spaced pyramid of images and then extract Dense SIFT feature for each scaled and smoothed (by a Gaussian filter) image. Using this method with a step size of 5, 250 clusters and an MLP Classifier, we achieved 62.6% accuracy.

4.4 SURF with BoVW

The SURF method is an algorithm for local similarity invariant representation, the main motivation for which was to speed up the execution time of SIFT.

Fist interest points (keypoints) on the images were detected by using a Hessian Matrix approximation [23]. In feature description, the keypoints are first used to find a reproducible orientation, and then ,using the chosen orientation, a square region is designed to describe the descriptors.

The SURF feature were built using the computer vision toolbox in MATLAB, and a visual vocabulary was built in a similar manner to run 2. Finally, the MATLAB statistics and machine learning toolbox was used to train a linear SVM classifier achieving 59.6% on the validation dataset.

4.5 Fisher Vectors

Fisher vectors (FV) [20] are yet another feature representation algorithm derived from the fisher kernel [21]. FV are an approximation to and improved version of the fisher kernel which is a generative model used to retrieve semantic information from documents. The reason we have decided to experiment with FV is that according to the research literature [10], FV outperforms significantly BoVW and also the Vector of Locally Aggregated Descriptors (VLAD) [9] which is an extended version of FV.

The advantage of the FV is that they keep higher order statistics from the features creating better representations. BoVW suffer from sparsity and high dimensionality, while FV use Gaussian mixture models (GMMs) in order to represent the codebook as a probabilistic model. To retrieve the FV we retrieve the SIFT features first from the images. The distribution of the features is modelled using GMM with 64, 128 and 256 Gaussians respectively. The FV utilise the posterior probabilities per local feature computing the covariance deviation and the means from each feature per each component of the GMM. In the last step the computed FV are fed into an SVM classifier. Table 4 illustrates the results obtained from the different GMM-FV models. The best results retrieved using 64 Gaussians yield 67.24% accuracy.

5 Conclusion

In this work we progressively increased feature and classifier complexity to do image classification. We started image classification by a applying standard classifier to simply generated features and progressed to state-of-the-art algorithms. Run 1 was developed by Yanislav. Run 2 was mainly developed by Matthew and Brent with help and additions from Pier and Yanislav. Run 3 was developed by everyone. Dimitris worked on the classifiers using SIFT and Fisher vectors. Matthew and Brent developed a set of classifiers to test their performance with pre-trained CNN features. Yanislav and Pier researched transfer learning and augmentation topics and Yanislav developed the CNN ensemble. Pier developed classifiers using HOG and SURF features and added Gaussian pyramids to the SIFT implementation from Dimitris.

References

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

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017.

[4] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005.

[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] PyTorch Documentation. Torchvision models.

[7] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR 2010 - 23rd IEEE Conference on Computer Vision & Pattern Recognition*, pages 3304–3311, San Francisco, United States, June 2010. IEEE Computer Society.

[10] Hervé Jégou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Pérez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, September 2012.

[11] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[12] Talia Konkle, Timothy F. Brady, George A. Alvarez, and Aude Oliva. Scene memory is more detailed than you think: The role of categories in visual long-term memory. *Psychological Science*, 21(11):1551–1556, 2010. PMID: 20921574.

[13] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[14] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[15] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Comput. Vision*, 43(1):29–44, June 2001.

[16] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[17] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[18] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, New York, NY, USA, 2016. ACM.

[19] 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.

[20] Florent Perronnin and Christopher R. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*. IEEE Computer Society, 2007.

[21] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *Int. J. Comput. Vision*, 105(3):222–245, December 2013.

[22] scikit-learn Documentation. Comparison of the k-means and minibatchkmeans clustering algorithms.

[23] David F Shanno and Kang-Hoh Phua. Matrix conditioning and nonlinear optimization. *Mathematical Programming*, 14(1):149–160, 1978.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[25] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, November 2008.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[27] Ricardo Vilalta, Christophe Giraud-Carrier, Pavel Brazdil, and Carlos Soares. *Inductive Transfer*, pages 545–548. Springer US, Boston, MA, 2010.

[28] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[29] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc., 2014.

[30] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 487–495, Cambridge, MA, USA, 2014. MIT Press.

Appendix A Run 2 Extras

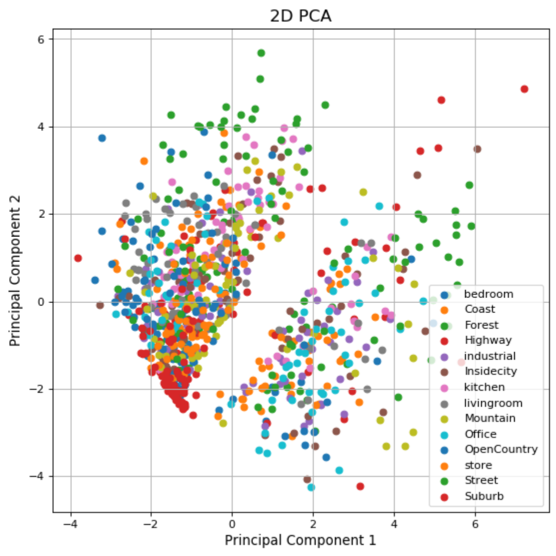
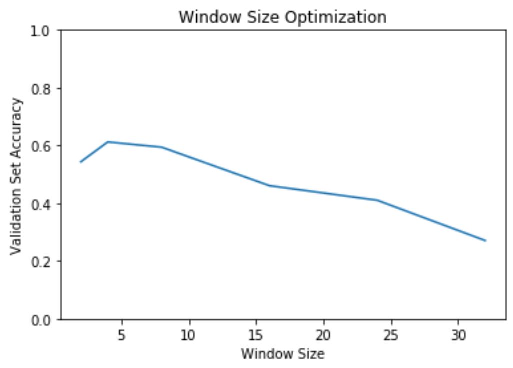
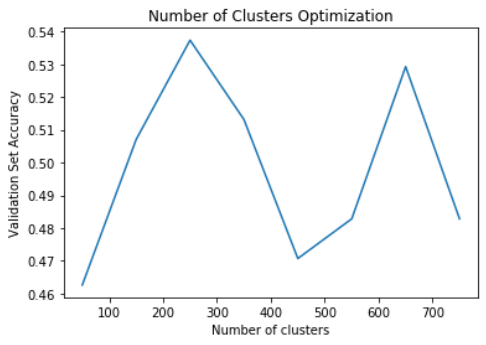


Figure 1: PCA 2D Representation of training feature space extracted using patches



(a) Varying window size



(b) Varying cluster size

Figure 2: Validation accuracy varying window and cluster size

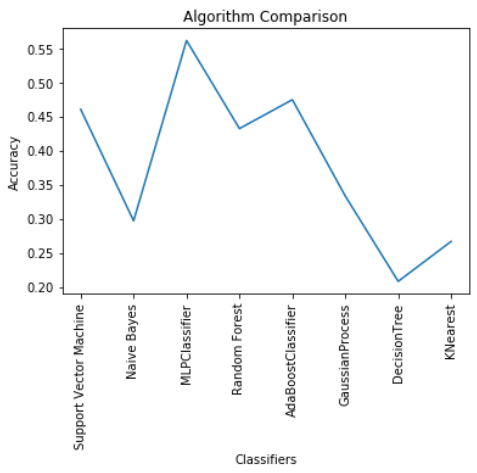
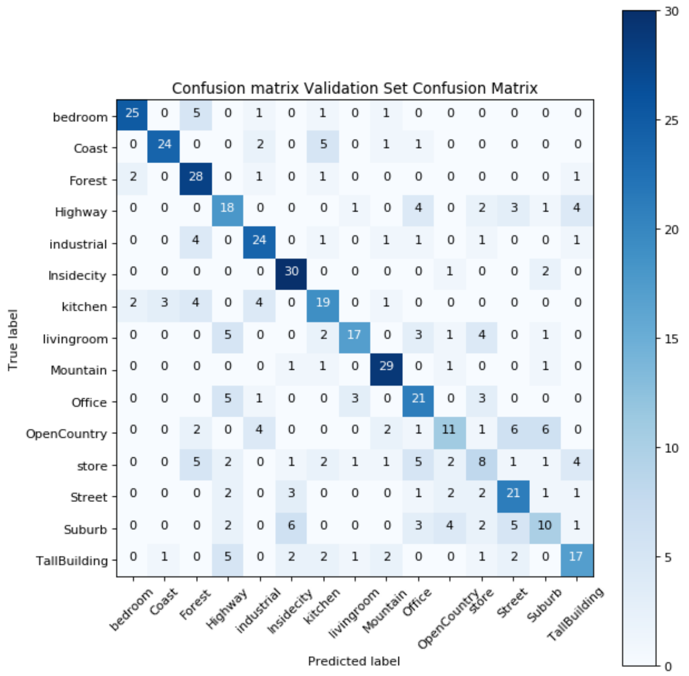


Figure 3: Validation Accuracy comparison using different algorithms

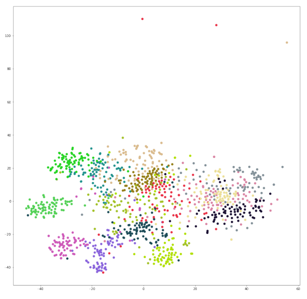


True label \ Predicted label	bedroom	Coast	Forest	Highway	industrial	Insidecity	kitchen	livingroom	Mountain	Office	OpenCountry	store	Street	Suburb	TallBuilding
bedroom	25	0	5	0	1	0	1	0	1	0	0	0	0	0	0
Coast	0	24	0	0	2	0	5	0	1	1	0	0	0	0	0
Forest	2	0	28	0	1	0	1	0	0	0	0	0	0	0	1
Highway	0	0	0	18	0	0	0	1	0	4	0	2	3	1	4
industrial	0	0	4	0	24	0	1	0	1	1	0	1	0	0	1
Insidecity	0	0	0	0	0	30	0	0	0	0	1	0	0	2	0
kitchen	2	3	4	0	4	0	19	0	1	0	0	0	0	0	0
livingroom	0	0	0	5	0	0	2	17	0	3	1	4	0	1	0
Mountain	0	0	0	0	0	1	1	0	29	0	1	0	0	1	0
Office	0	0	0	5	1	0	0	3	0	21	0	3	0	0	0
OpenCountry	0	0	2	0	4	0	0	0	2	1	11	1	6	6	0
store	0	0	5	2	0	1	2	1	1	5	2	8	1	1	4
Street	0	0	0	2	0	3	0	0	0	1	2	2	21	1	1
Suburb	0	0	0	2	0	6	0	0	0	3	4	2	5	10	1
TallBuilding	0	1	0	5	0	2	2	1	2	0	0	1	2	0	17

Figure 4: Run 2 Confusion Matrix

Appendix B Run 3 Extras

As an addition to this project, we decided to save the features extracted using transfer learning and to use them to train and test a variety of Machine Learning classifiers to see if it could have been possible to increase our overall prediction accuracy. As we can see from Table 6, classifiers which have been trained using the ResNet50 features are the ones which performed overall better. In fact, as shown in the t-SNe in Figure 5, ResNet 50 has more separated features compared to VGG19.


(a) Resnet features.

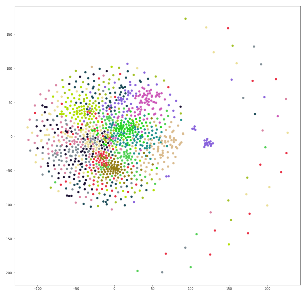

(b) VGG features.

Figure 5: Projected features on 2D using T-SNE.

	VGG19	ResNet50
Nearest Neighbors	25%	58.66%
Linear SVM	**87.66%**	**87.66%**
RBF SVM	80.66%	87.33%
Gaussian Process	7%	7%
Decision Tree	43.33%	53.66%
Random Forest	42%	40.66%
Neural Net	79.33%	86.66%
AdaBoost	19.66%	20.66%
XGBClassifier	81%	85%
GradientBoostingClassifier	73%	78%
BaggingClassifier	65.33%	73.33%

Table 2: Results from a number of different classifiers on VGG19 and ResNet50 features.

clusters/classifier	SIFT s=5	SIFT s=10
500/SVM	51.13%	57.06%
500/RF	**59.53%**	57.57%
1000/SVM	32.85%	57.06%
1000/RF	57.86%	56.65%

Table 3: Table results for sift for step 5 and 10 pooling features.

N-Gaussians/Classifier	Fisher/GMM
64/SVM	**67.24%**
128/SVM	65.43%
256/SVM	62.08%

Table 4: Table results for Fisher kernels with GMM.