CSC411: Machine Learning
Assignment 3: Image Scene Prediction
Jacob Ritchie (999771983)
Bill Zhao (999721127)
Kaggle Team: Swagalicious

1.Introduction

The challenge given in the assignment was to classify images into 8 general categories based on their features.
Our intuition was to make use of a convolutional neural network (CNN), because these are the technology that currently perform best on similar image classification benchmarks (for example, the majority of the categories in the 2015 ImageNet competition were won by CNN models or ensembles made up of CNN models).
We attempted to train our own CNN on the given training data, but found we were unable to generalize to the test set - to proceed, we made use of a pre-trained CNN (TensorFlow implementation of the Inception-v3 architecture), fine-tuning the network's final layer to match our specific dataset. We also considered a manual feature vector extraction scheme (GIST) which could be used as an input to a number of different classifiers - we considered algorithms such as AdaBoost, and support vector machines (SVMs) with various kernels.

2.Description of Submitted Solution

Our submitted solution made use of transfer learning to adjust a large pre-trained convolutional neural network classifier to fit our specific design problem. The pre-trained neural network that was chosen was the Inception-v3 convolutional neural network developed by Google. This network was trained on the 2012 ImageNet Competition data, which consists of 1.2 million training images (a process which can take weeks). The model, seen in figure 1 below, consists of a large number of convolutional layers (which learn to detect translation-invariant image features), and pooling layers which downsample the representation to control overfitting. The output of the network is a fully connected-layer with dropout regularization, leading into a final softmax layer.

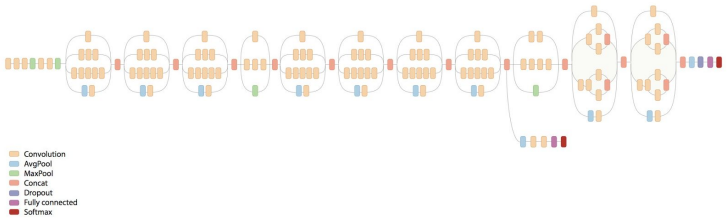

Figure 1: Inception-v3 CNN architecture

To retrain the neural network to our specific problem, we modified the softmax layer to have 8 outputs rather than the 1000 used for the ImageNet competition. Then, we loaded the pre-trained weights for all layers except the final softmax layer, and used batch gradient descent to find the final layer weights. We used a RMSprop, an adaptive back propagation method that decreases the learning rate over the course of training. The retraining process took place over 4000 training epochs on a c4.4xlarge AWS instance.

The following hyperparameters were used in the retraining phase. A small-scale hyperparameter search was conducted (varying each parameter over a linear search space with 3 values each), but the default settings given by TensorFlow for the retraining process proved effective for our problem, and we did not change them significantly.

Table 1: Inception-v3 Retraining Hyperparameters

RMS propagation decay	0.9	Initial learning rate	0.1
Momentum	0.9	Learning rate decay factor	0.14
RMS propagation epsilon	1.5	Epochs per decay	30

Though we made use of data augmentation and class-balancing strategies for our other attempts at solving the problem, neither of these proved necessary for the pre-trained net.

3.Results and Comparison to Alternative Solutions

The results of the three solutions that we considered are as follows:

Table 2: Comparison of Model Performance

Model	Best Validation Set Accuracy	Best Public Test Set Accuracy
Pre-trained CNN	85.2%	80.2%
CNN	54 (± 3)%	27.2%
GIST + RBF-kernel SVM	52 (± 2)%	26.0%

Using the following accuracy metric:

$$a = \sum_i \frac{\mathrm{I}[\mathrm{pred}_i == \mathrm{label}_i]}{N_{\mathrm{sample}}}$$

For the CNN and SVM models, five-fold cross-validation was used to estimate the model performance on the training data. The error in the mean was calculated from the five accuracy scores. Because of the long time interval necessary to retrain the Inception network, cross-validation was not feasible. Instead, the training data was split into a training set of 6125 images and a validation set of 875 images and this was used to evaluate the model's validation performance.

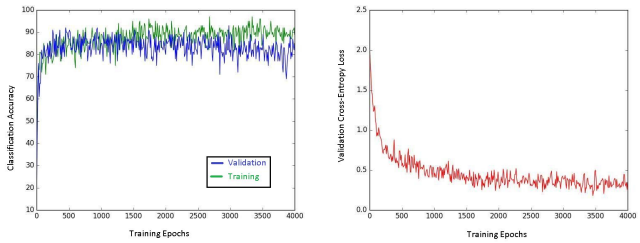

Figure 2: Inception-v3 classification performance during retraining

We can see that the pre-trained CNN's validation set accuracy is higher than its test set performance by only a small margin. This means that it generalizes well to the remainder of the data. The other two methods that we attempted, though they achieved reasonable validation set performance, failed to generalize to the test set data. Our hypothesis is that due to the nature of the training set (which has a very low number of images and suffers from a large class imbalance - see figure below), even with a well-selected set of hyperparameters the training set is insufficient for generalization, especially since the models we selected are prone to overfitting.

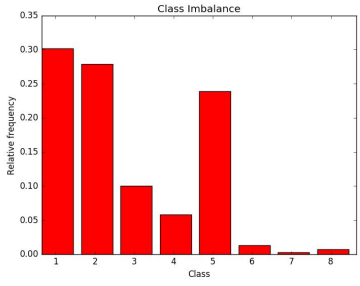

Figure 3: Training Set Class Frequencies

The high performance of the pre-trained CNN is due partly to its complex and highly-optimized design, but also to the fact has been trained over a long period on a huge dataset, allowing it to perform well on a wide range of image classification tasks.

The baseline solution for this problem used the GIST feature extraction library, and then a simple k-nearest-neighbour classifier. This achieves baseline performance on the public test set of 44.2%. GIST is a form of feature extraction, which condenses each 128x128x3 image into a single 1x590 feature vector through convolution with Gabor filters (see figure 4).

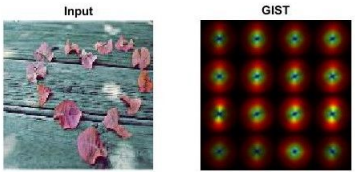

Figure 4: Training image and visual representation of GIST descriptor

Using the MATLAB implementation presented in the assignment, GIST features were extracted. They were then normalized before being used as the input to a support vector machine. (Since SVMs are not invariant to the size of the input features, this is a necessary step to achieve correct classification).

Several different kernel functions were evaluated, namely the linear, polynomial, sigmoid and radial basis function kernels. Initial testing with the default hyperparameters indicated that the polynomial and sigmoid kernels were not useful (validation set accuracies were no better than random guessing). A hyperparameter search was conducted for both the linear and RBF kernel SVM models. An exponential search space for the hyperparameter C (the model's penalty factor) was evaluated, the results of which are shown in figure 5.

As a result of this search, we found that the optimal values of C were 1 for the RBF kernel and 0.01 for the linear kernel, which resulted in cross-validation accuracies of 47 (± 2)% and 45 (± 1)% respectively.

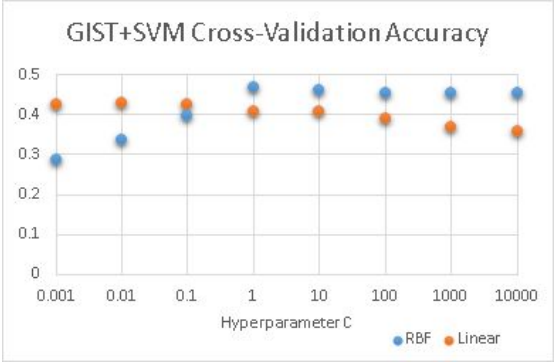

Figure 5: SVM Hyperparameter Sweep Results

However, the results of this model on the public test set (26.2% accuracy) were rather unimpressive. In fact, this performance is lower than simply assigning all images to the most numerous class (~30% accuracy). This was likely the result of an inability of the model to generalize from the training data to the validation data, due to the small size of the data set. To remedy this, we considered two strategies (class balancing and data augmentation), which are discussed below.

The other alternative model that we considered was a custom-built CNN. It was determined that the model architecture (number of convolutional layers, number of fully-connected layers) was the hyperparameter that would contribute most strongly to this solution's performance. We evaluated several different model architectures. We varied the number of convolutional / max-pooling layer pairs between 1 and 4. It was found, as expected, that adding convolutional layers increased the maximum validation accuracy of the model. We also found that the smaller networks started to overfit to the dataset before we reached 200 training epochs,

while the larger network showed none of the typical effects of overfitting (validation cross-entropy did not increase).

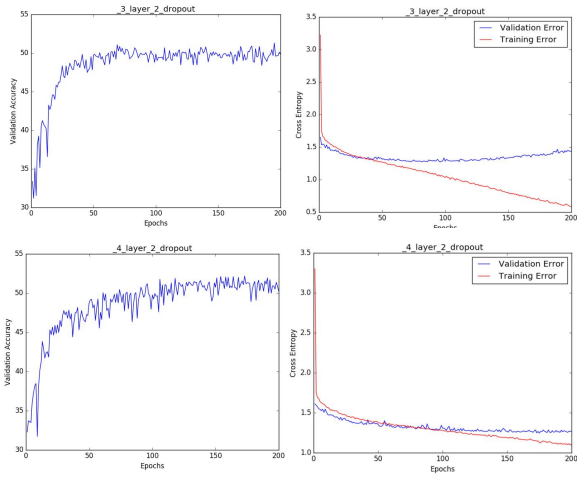

Figure 6: Effect of additional convolutional + max pooling layers

The model that was selected was a convolutional neural network consisting of 7 layers (4 convolutional / max-pooling pairs). The model architecture is shown in the following table.

Table 3: Selected CNN architecture

Type	Patch-Size / Stride or remarks	Input-Size
Conv	5/2	128x128x3
Max-Pool	2/2	128x128x5
Conv	2/2	64x64x5
Max-Pool	2/2	64x64x2
Conv	2/2	32x32x2
Max-Pool	2/2	32x32x2
Conv	2/2	16x16x2
Max-Pool	2/2	8x8x2
Dropout	rate=0.5	8x8x2
Fully-connected	-	1x1x256
Dropout	rate=0.5	1x1x256
Softmax	Classifier	1x1x8

During our initial investigation of different model architectures, the depth and filter size were fixed at 128 filters of size 4x4 for the initial convolutional layer and filters of size 2x2 were used for the other convolutional layers, with the pool size fixed at 2x2. This was motivated by the finding in the literature that multiple layers of small filters outperform fewer layers with larger filters.

Once a good choice for the model architecture had been determined, the additional hyperparameters, learning rate, momentum, number of training epochs and batch size were varied in a second parameter sweep (3 values of each parameter were considered).

We found the best hyperparameters for our architecture to be the following. These are consistent with typical CNN hyperparameters that we found in the literature (please see the references section).

Table 4: Selected CNN Hyperparameters

Momentum	Learning Rate	# Epochs	Batch Size
0.9	0.001	200	100

The effectiveness of replacing the ReLU activation function with a sigmoid activation function was evaluated, but that was found to decrease classification accuracy by 6.2% over the same number of training epochs

We also examined several regularization methods. Our initial model architectures used 2 dropout layers at the end of the network for regularization purposes. When we removed these layers and added L1 regularization to our gradient descent function (which directly penalizes large weights), the cross-validation accuracy suffered, dropping from 51% to 45%. We also attempted reducing the number of dropout layers to examine whether they were harming network performance, but found that removal of dropout regularization resulted in dramatic overfitting.

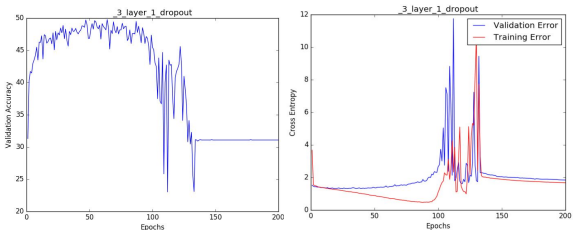

Figure 7: Overfitting following dropout layer removal

Though we performed many experiments, we did not succeed at generalizing our validation set performance to the test set - our maximum test set performance for the CNN was only marginally better than that of the GIST + RBF-SVM solution. We followed the following two strategies in an attempt to remedy this problem.

Because of the very small size of the training set considering the complexity of the problem, data augmentation was used to increase the size of the training set and increase invariance to small changes in image features. Using MATLAB, horizontal reflections, random rotations of up to 45 degrees and random scaling of the image color channels were generated for all 7000 images in the training set (resulting in an augmented training set size of 28,000).

These were then used directly as inputs to the CNN, and GIST feature vectors were generated for the augmented training set to use as inputs to the SVM model. Training on the augmented data set was observed to have a small positive effect on the cross-validation accuracies for both models (a 5% increase for the CNN and 3% for the SVM), but considering their low public test performance seemed unlikely that it would result in significant enough improvement to beat the baseline solution.

Figure 8: Example of data augmentation

We also found that the class-imbalance of the training set (seen in figure 2) was magnified by the predictions of our model. We found that the three most-frequently occurring classes were being predicted an overwhelming amount of the time, (93.2% of the predictions made by our RBF-kernel SVM classifier were for the three majority classes) meaning that the other 5 classes, which make up more than 10% of the dataset, were underrepresented in our predictions.

(We also consistently saw the label "2-indoor" being over-predicted across all 3 models. This is likely due to the fact that many of the images that had the other labels were also taken indoors. The overlapping nature of the class labels made the classification problem more challenging.)

To correct this class imbalance, oversampling was used. The CNN's batch training iterator was modified to examine each of the images from the less numerous classes twice. Because of the already small size of the training set, undersampling (discarding training data from the more numerous classes) seemed unlikely to produce good result. The scikit-learn SVM implementation which we employed contains a class-balancing parameter, which allowed the effective value of C to be increased / reduced for each class (effectively resulting in a higher penalty for misclassifying examples from underrepresented classes). By tuning this parameter until the proportions of predicted classes on the validation set matched the

proportion from the dataset, we improved the cross-validation accuracy of the rbf-SVM to 52%, which occurred at the following set of class weights.

Table 5: SVM Class Balancing Weights

1	0.5
2	0.5
3	1
4	1.5
5	0.5
6	4
7	8
8	4

Both the techniques that we used to cope with the limitations of the training set resulted in measurable performance increases on the cross-validation accuracy, for both of our classifiers. However, these gains were insufficient to allow us to generalize to the test set data and beat the baseline accuracy of the GIST+kNN classifier.

The process of generalization was much easier for the Inception-v3 net, because of its highly-optimized architecture and use of additional training examples, which is what we were ultimately forced to rely on to achieve the desired classification performance. This underscores the powerful effects of model selection and the importance of a good training data set, which were repeatedly emphasized in this course.

Bonus Problem: Multi-label classification

There are 24 bonus classes,we begin by training 24 binary classifiers to detect whether or not each label is applicable to each image. We opted to use the GIST + rbf-kernel SVM method above, because of its speed relative to the other methods we investigated.

We also create a 24x24 co-occurrence matrix from the training data to keep track of the relationships between classes. The equation for the co-occurrence matrix is as follows (where x_n is the nth training example):

$$[C_{ij}] = \frac{\sum_n I[has_i(x_n) \cap has_j(x_n)]}{\sum_n I[has_i(x_n)]} \approx \frac{P(i \cap j)}{P(j)} = P(i|j)$$

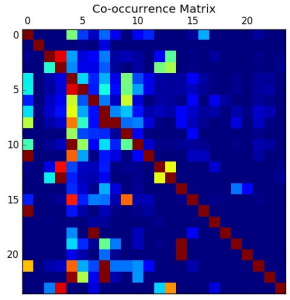

Figure 9: Co-occurrence Matrix for Training Data

To incorporate the results of the co-occurrence matrix into our data, we make use of Bayes' rule. We configure the SVM classifier to assign initial labels and return the probabilities of each label. Then, using each label i that we have assigned, we see if the posterior probability of every other label j (given our observation of i) is higher than some threshold using Bayes rule, (where we approximate P(j|i) using the co-occurrence matrix, j) and update the labels accordingly.

$$P(j|i) = \frac{P(i|j)P(j)}{P(i)} \approx \frac{C_{ij}P(j)}{P(i)}$$

References

The architecture of the pre-trained neural network comes from the following paper:

[1] Christian Szegedy, Vincent Vanhoucke, et. al. “Rethinking the Inception Architecture for Computer Vision”

To implement the transfer learning process we made use of the example code from the Tensorflow github repository:

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining

In our project we made use of the Theano and Lasagne libraries to implement our scratch-built convolutional neural network

[2] Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions”.
[3] Sander Dieleman et. al. "Lasagne: First release."

Choice of convolutional neural net architecture was informed by the following analysis:

[4] Ken Chatfield, Karen Simonyan, “Delving Deep into Convolutional Networks”

The dropout regularization technique was proposed in 2014 by machine learning researchers at the University of Toronto.

[5] Srivastava, Nitish, et.al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from overfitting". Journal of Machine Learning Research.

Our SVM model was implemented using the scikit-learn python library

http://scikit-learn.org/

The GIST feature descriptor comes from the following publication:

[6] Modeling the shape of the scene: a holistic representation of the spatial envelope Aude Oliva, Antonio Torralba International Journal of Computer Vision, Vol. 42(3): 145-175, 2001

The GIST MATLAB implementation we used is available here:

http://people.csail.mit.edu/torralba/code/spatialenvelope/