Akila Mahgammulla Gamage 999847395
Abhijit Kambalapally 999864703

CSC411 Introduction to Machine Learning Assignment 3

1.0 Introduction

Convolutional Neural Networks and Support Vector Machines were used as our approaches because both of them are proven to work well in image classification and seems to stand out from the alternatives as evaluated below:

Alternatives not chosen:

Linear Regression: It cannot perform multi-class classification.

Logistic Regression: It can potentially perform multi-class classification using multinomial logistic regression but still expects features to have linear thresholds. Since this is not the case for the provided images, it will cause most common classes to dominate, thus it will provide suboptimal results.

K-nearest neighbours: Relies too much on the distance between features, and since the images provided seems to have the similar features but at different locations of the image, it will yield poor results. This is used as the baseline.

Decision Trees: Even though it provides quick results, the fact that decision boundaries are axis aligned, can lead to poor results in complex problems like image classification that involve a large number of features.

Vanilla Neural Networks: As evident from Assignment 2, it was clear that Convolutional Neural networks outperformed Vanilla Neural Networks approach due to the convolutional step which generates stronger features that are more invariant to image distortion and positioning (this also gives us more freedom to preprocess the images).

Alternatives chosen:

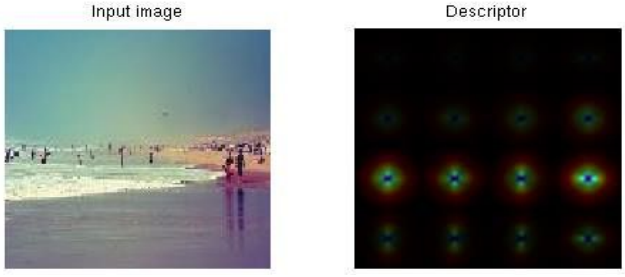
Support Vector Machines: Provides robust results for even small datasets, allows flexible hypothesis, uses soft-margin extensions allow mis-classified examples to still be correctly classified and provides faster convergence time compared to other more complex models.

Convolutional Neural Networks: Generates stronger, more invariant features through convolutional filtering. This allows sharing of weights across multiple features which can leads to better representation of features than the alternative models and faster convergence the conventional neural networks when there's a large number of features.

2.0 Discussion of Approaches Implemented

2.1 GIST-KNN

The baseline used to evaluate the models we implemented was a KNN model with takes the output of the GIST descriptor as input. Intuitively, GIST summarizes the gradient information (scales and orientations) for different parts of an image, which

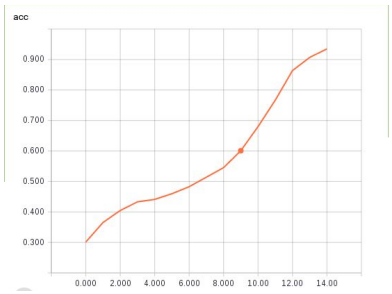


provides a bough description (the gist) of the scene.
GIST hyper-parameters used were the default = [8, 8, 8, 8] and the K for KNN was chosen to be 50 through cross-validation over the training dataset. The values tried for K = 1, 2, 5, 10, 50, 100, 200. A validation accuracy of 44.22% was obtained.

2.2 Convolutional Neural Networks

Our initial CNN consisted of 2 hidden layers with Relu activation and the final later with Softmax activation with max pooling in between. The baseline parameters used were **-** Learning Rate: 0.01, Batch size: 32, Epochs: 15, Decay: 1e-6, Momentum: 0.9 which gives a 44% accuracy as seen on the right. Next, various hyperparameters were optimized and results are displayed below.

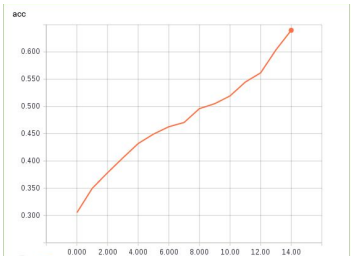
Validation Accuracy per Epoch

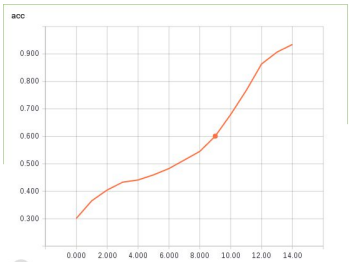


Training Accuracy per Epoch

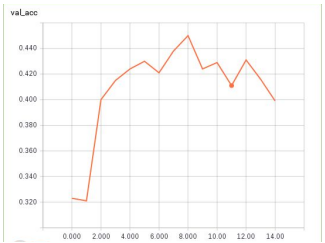
2.2.1 Varying Learning Rate: Chose 0.002, since higher learning rates results in high fluctuations in validation accuracy.

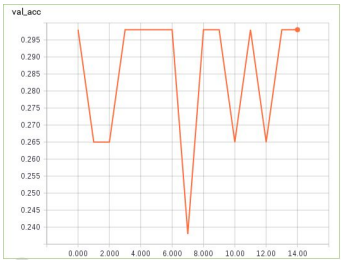
0.002 **0.01** **0.1**





Training Accuracy per Epoch

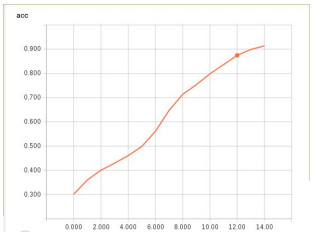


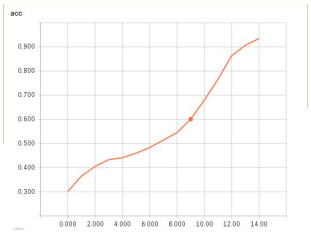


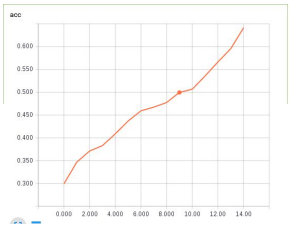
Validation Accuracy per Epoch

2.2.2 Varying Batch Size: Chose 64, since validation accuracy seems to fluctuate for smaller sizes, due to small sample sizes used for gradient descent become more prone to bias and noise.

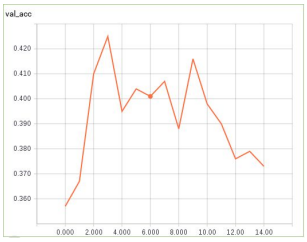
16 **32** **64**

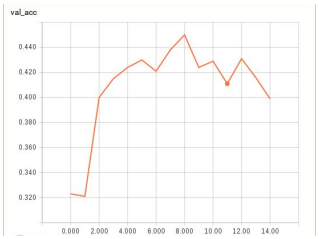


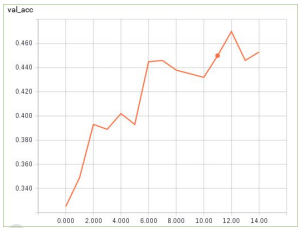




Training Accuracy per Epoch



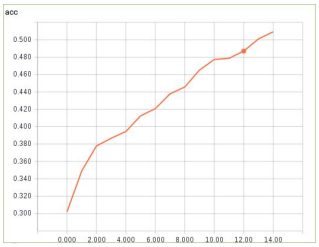


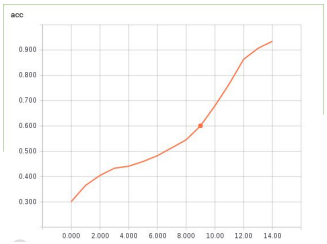


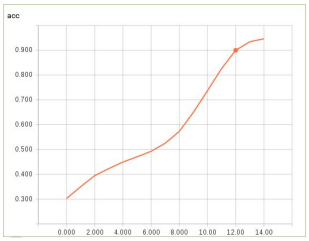
Validation Accuracy per Epoch

2.2.3 Varying Momentum: Chose 0.9, since when the momentum is increased, the network converges much faster, but when it's too high it converges too fast, which leads to overfitting.

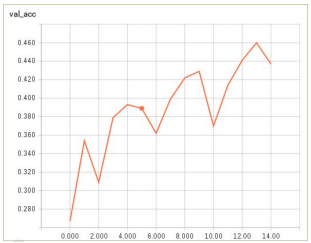
0.6 **0.9** **1.2**

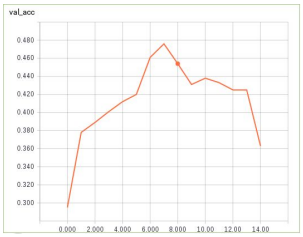






Training Accuracy per Epoch





Validation Accuracy per Epoch

2.2.4 Final version

Our final version was a 7 layer NN with relu activation and final layer with softmax activation with max pooling in between activations. The hyperparameters are Learning Rate: 0.002, Batch size: 64, Epochs: 15, Decay: 1e-6. We further enhanced the model by adding drop out layers after pooling and also generated additional images to make the model more robust. Some of these image features were rotation, shear, width shift, height shift, and horizontal flipping.

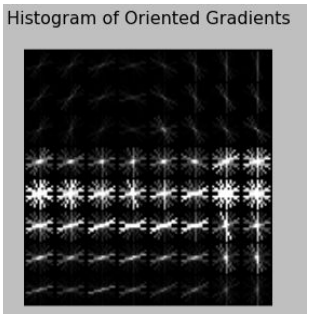
2.3 Support Vector Machines

The following section includes our empirical results from our implementation process and discusses our results.

2.3.1 Preprocessing

Unlike convolutional neural networks which is capable of extracting input features from any type of inputs like images, we found that SVMs require much more feature engineering. We tried various feature extractors such as

- Simple subsampling: created smaller size (16x16) images for each input image so that the resized images are less sensitive to relative position, alignment etc.
- SIFT (scale invariant feature transform) [1]: extracts keypoints from the images where each image may be described using different number of keypoints. However, SVM requires same of inputs, therefore, we had to run kmeans to find the most significant N features and input this to our SVM. However, we did not see a very big improvement.
- HOG (History of Oriented Gradient [2]: accumulates gradient information, edges and orientation histograms which are invariant to illumination, shadowing, contrast etc. Overall, this descriptor gave us the best improvement in accuracy. An example from the training data has been plotted below. in image format. Its vector representation is passed as the input to the SVM.



2.3.2 Picking the kernel

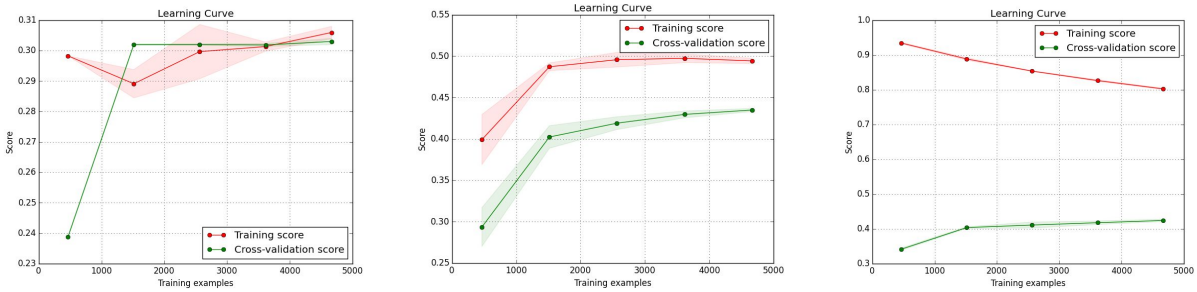
Three kernels were tried and their outputs have been given below. Initially, before doing any preprocessing on the images, the linear kernel performed the best since there were a large number of input dimensions (128x128x3) >> number of samples. Therefore, the model did not needed to be mapped to an even higher dimension and the linear model was good enough. However, after preprocessing, our input features reduced to around 2000 features with HOG (<< number of samples), so using a nonlinear kernel like the RBF (radial basis function) expectedly performed the best.

Kernel	Linear	Polynomial (degree 3-8)	Radial Basis Function
Validation Accuracy	38%	~33-35%	40%

2.3.3 Changing hyper-parameter C and gamma

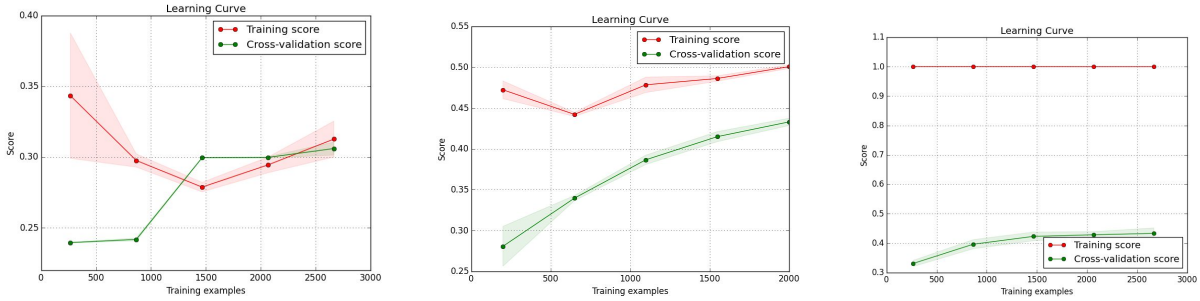
There were two hyperparameters for the RBF kernel were C and gamma. C is the penalty parameter for the error term and essentially controls the influence of each support vector,(provides a soft margin). A low C seems to make the decision surface smooth thus, allowing for more error, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. A value 10 was found to be the optimum.

C=0.1 C= 10 C= 100



Gamma is the kernel coefficient, which controls the amount of influence (radius) and variance a support vector has over the decision boundary. A high gamma leads to a high bias and lower variance and vice versa as seen in the plot below. A value of 0.01 was found to be optimum.

gamma = 0.001 gamma=0.01 gamma = 1



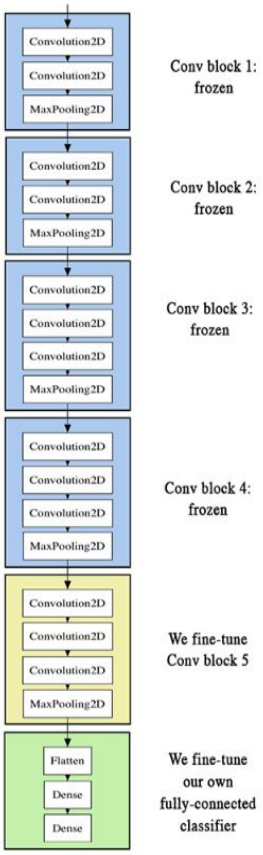
2.3.4 Class Weights

We also took in considered the uneven distribution of the classes in our training data set. Some classes had many more examples than others. To account for this in the SVM, we set the parameter C of class i to class_weight[i]*C for the SVC. So the class with few examples will not be dominated by the other classes.

3.0 Submitted Solution

The final solution submitted was based on a pre-trained model that was trained on the ImageNet dataset [3]. We utilized a pre-trained network, VGG16 and used its corresponding

pre-trained weights. As for training only the last convolutional block is trained using the training images. This makes sense, since as we go deeper into a CNN the features we find become less abstract. Thus, by using pre trained weights that has been trained using large amount of examples for recognizing these abstract features we avoid overfitting the network to our own data. This also allowed us to run the network much faster, and chose appropriate hyper parameters to further improve our classification.



Final Hyper Parameters:

- Batch size = 32
- Learning rate = 1e-3
- Momentum = 0.9

In addition the output block was added to include a Flatten layer followed by 2 additional fully connected two Dense layers.

3.1 Evaluation of the three approaches

Model	GIST + KNN	HOG + RBF SVM	6 layer CNN	Pre-trained CNN
Accuracy	44%	46.3%	56.5%	60.7%

The above results show that GIST + KNN was outperformed by all other approaches. This outcome was predicted in the introduction, where it was mentioned that KNN relies too much on the location of features and the fact that the data showed a varying distribution of features for images of the same class. Even though HOG + RBF SVM shows a rather low accuracy it had the top speed of all the training models. This advantage can be significant when adjusting hyper parameters and when time for a solution becomes a factor when you have a large number of training samples. The results also show that the pre-trained CNN model performed the best. This is because the provided training data was very biased towards specific classes and was not large enough to provide adequate amount of features. So by using the pre trained weights, it was possible to ignore learning more abstract features by focusing on finding specific features that defined the classes.

4.0 Conclusion

Our method for pre-trained CNN performed adequately given the skewed data distribution. However, it was evident that there is room for improvement in our models. A few ways we can improve our performance may be through the use of a deeper neural network, with many more hidden layers. We could also incorporate other preprocessing techniques before insert our training data into the models. The overall performance may have been improved by using ensemble methods and furthermore the possibility of using an SVM as the final layer of a Convolutional Neural Network.

References

[1]
http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[2]
http://scikit-image.org/docs/dev/auto_examples/plot_hog.html

[3]
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Resources Used

Convolutional Neural Net

- https://keras.io/
- https://www.tensorflow.org/
- AWS GPU

Support Vector Machine

- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

HOG feature extraction

- http://scikit-image.org/docs/dev/auto_examples/plot_hog.html

SIFT feature extraction

- http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html