ASSIGNMENT 3
Kaggle Team - Los Perdedores

Juliana De La Vega Fernandez
j.delavegafernandez@mail.utoronto.ca
id:1003092468
Chocolatina

Rafael Pedrosa Lacerda de Melo
Rafael.pedrosalacerdademelo@mail.utoronto.ca
id: 1003633017
Lacerda

I. Introduction

An effective strategy for image classification is feature extraction. It has the goal of generating features such that the detection of a pattern of features in an image will distinguish a particular class from the rest. Transfer learning has proven useful in feature extraction, especially in problems where training and rebuilding a model is expensive. Transfer learning is described as the "improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned" [1]. We will attempt to use the concept of transfer learning to extract features from a previously trained ResNet50, used for image classification, and train these features using a Support Vector Machine.

A quick glance at the provided dataset reveals that class frequencies are imbalanced, and classifiers can be skewed towards classes with higher frequencies. To account for this, we have used data augmentation techniques to rebalance class frequencies. Rotations in 90 degree increments, flips and jittering were performed. To address the issue of overfitting, K-Fold cross-validation was used. Due to the heavy demand for computational power required by the classifiers used, we have adopted the parsimony principle and compromised on two folds. As a comparison to the proposed model, a separate K-Nearest Neighbours approach was also trained to attempt the classification of the images. The results using the combination of transfer learning from a ResNet50, and an SVM for training are promising.

II. Submission

A. Description

Neural networks can be built using two principles: modularity and residual learning. Modularity refers to having a small network which is repeated to increase depth, while residual learning is what ensures that something new is learned from a new layer added. ResNet50 is a deep residual learning network developed by Microsoft Research. Its framework is deeper than previous residual networks. Layers in this network are formulated as residual learning functions regarding the layer's inputs [2]. This network received first place in ILSVRC 2015 image classification. Its implementation for ImageNet first resizes an image and samples it by cropping it by 224x224 with the pixel per pixel mean subtracted. Afterwards, standard colour augmentation. After each convolution, the batches are normalized [2]. In our implementation, we used a previously trained model where the weights had been initialized and all the plain and residual nets had been trained [3]. The authors describe that the learning rate starts at 0.1 and is divided by 10 when the error plateaus [2]. They also report using a weight decay of 0.0001 and a momentum of 0.9.

The fact that ResNet is a Deep Residual Network allows it to be more generalizable, as it will be trained more efficiently, gaining new data from each layer and facilitating learning. Additionally, vanishing gradients are avoided using ReLu activations since this function

behaves linearly above 0[4]. Therefore, we chose to use ResNet50 over other pretrained CNN's. We concluded that using the residual learning would prove useful to extract meaningful features from any image class, even if it was not contained in the ImageNet classes. In order to prove this, we implemented Google's VGG 19, and although the results went over the established baseline, there was more than a 10% difference in accuracy with the ResNet50 implementation over the validation set. This network has an architecture that is divided into five blocks which perform consecutive convolutions, and pass the data through a pooling layer at the end. After these five blocks, data is flattened and goes through three fully connected layers that perform two ReLu activations and a SoftMax activation to obtain predictions.

Conversely, ResNet's architecture displays modularity. Its first six layers modify the image using six layers of transformation which are: padding the image with zeros, performing a convolution, normalizing, calculating ReLu activation, and it performs a max pooling before the data goes through to another set of layers. The sets of layers that follow are the ones that display modularity. The data goes through a convolution block which performs convolutions, normalizations and activations with ReLu, and it has a convolutional layer at the shortcut connection layer. Consecutively, the data goes through an identity block, which only differs from the convolution block in that it does not contain a convolution layer at the shortcut connection layer. These two blocks are the modular part of ResNet, they are present if four blocks of layers, in which a convolution block is applied, followed by various identity blocks. Lastly, at the end of these blocks, the data goes through an average pooling layer, and by three fully connected layers at the top of the network.

Using Keras [4], ResNet50 was implemented. We did not include the three fully connected layers at the top of the network. This included the activation layer, which could be easily deactivated by setting the include_top flag to False. The benefit this would give is, rather than the classifications for the ImageNet classes for which the net was trained, we obtained the resultant 2048 array from the average pooling layer. This array will now be our feature vector that will be classified after training a SVM. This would mean that for each image, instead of having to deal with its original 128x128x3 features (pixels in RGB), we would only focus on the relevant features that ResNet extracted. The advantage of ResNet 2048 size array for each image is that every feature is different, and gathers information that is relevant towards classification. Using the arrays obtained, we trained a support vector machine to classify the data.

Support Vector Machine classification creates different planes in the feature space to divide the data. This dividing plane has a special characteristic, its dividing plane maximizes the margin between the different classes, thereby reducing the generalization error of the classifier [5]. The experiments performed to set the SVM's hyperparameters will be described in the Empirical Results section.

B. Data Processing

As was described in the introduction, data augmentation was performed to achieve a uniform distribution between the classes. After the transformations were performed, each class contained 2114 members. The data was organized into a .npz file to make the process of loading images easier. Additionally, using k-fold cross validation, the augmented dataset comprised by 16912 images was distributed into two batches. One of them was used as training, and the other as validation. To pass the images through the pretrained ResNet50, they had to be resized to 224x224.

C. Algorithms Used

For the k-fold cross validation, KFold from sklearn.model_selection was imported, the shuffle flag was set to true, and the number of splits was set to two. Each of these batches were saved with their corresponding class annotation.

Keras, tensorflow, imagenet_utils and scipy were used to obtain the max-pooling features from ResNet50. Each image in the training batch was resized to 224x224, expanded along one axis, passed through the ResNet. The resulting prediction was flattened and saved. This process was performed for the validation and test set as well.

Using the metrics and svm module from the sklearn library, the SVM classification model was trained. First the model was declared with its corresponding hyperparameters, and then fit with the training batch input, and its labels. Afterwards, we obtained the prediction for the variables using model.predict with the validation batch images. Using the metrics module, we obtained the classification report for each class and the confusion matrix. With these we could measure the Precision, Recall and F-score given the predictions on the validation set and the actual label on these images. Finally, we predicted the labels for the test data, and saved it to a .csv file.

III. Empirical Results

To decide on the hyperparameters for the SVM, we performed a series of experiments in which all hyperparameters were maintained fixed, except for the one that was being evaluated and proceeded to train the SVM with that changing parameter. This is how we fine-tuned the penalty parameter C of the error term, the kernel type, the degree of the polynomial kernel, and the shape of the decision function. As may be seen in Figure 1, six different values of C were evaluated, [1, 3, 6, 9, 11]. Increasing the C parameter improved the accuracy in 2%, and this was maintained through the higher values. For simplicity, C=6 was chosen as the best, given that it applied less penalty, and was not in the threshold for the increase in accuracy.

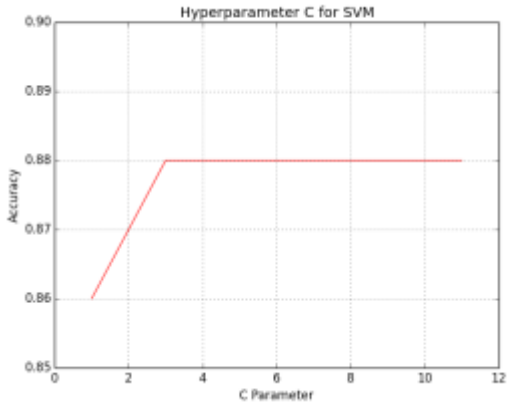

Figure 1. Penalty Parameter C vs Accuracy for SVM classification.

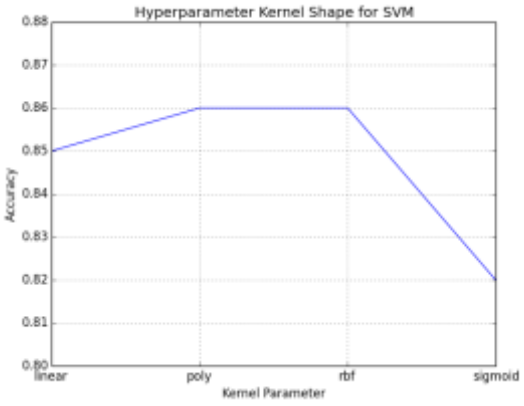

Figure 2. Kernel Shape vs Accuracy for SVM classification.

Figure 2. shows the evaluation of the four types of kernels that may be applied in an SVM, the best performance was seen while using the polynomial kernel, or the radial basis function. We chose the default that is used in sklearn, which is the radial basis function. Figure 3 displays the evaluation of different values for the degree of the polynomial function and its effect over the accuracy of the algorithm. The degrees evaluated were 3, 6, 9, 12, and 15. As may be observed in the figure, all the degrees evaluated displayed the same result, which is

why the default, 3, was chosen. Figure 4 shows the result of varying the decision function shape used in the SVM, once again, all the shapes evaluated displayed the same result, which is why the default for lib-svm, one-vs-one was selected.

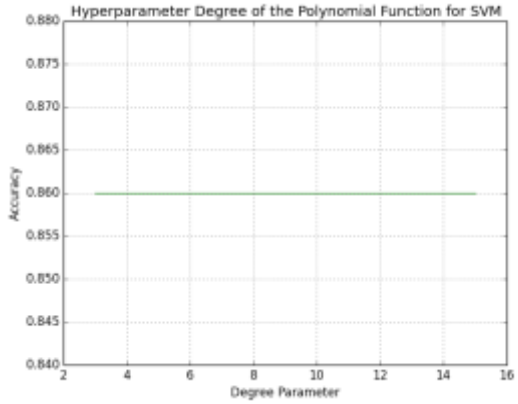

Figure 3. Degree of the polynomial vs Accuracy for SVM classification.

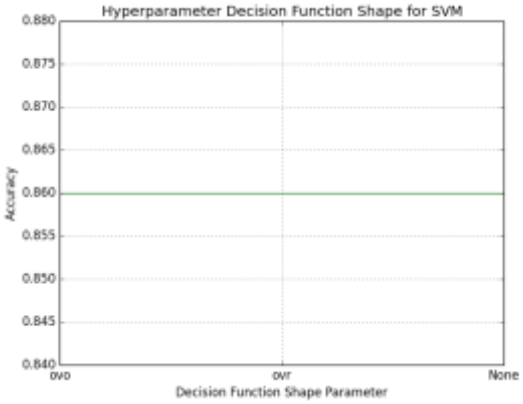

Figure 4. Decision function shape vs Accuracy for SVM classification.

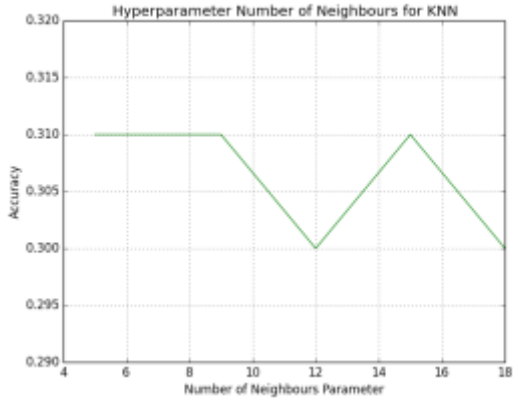

Figure 5. Number of Neighbours Parameter vs Accuracy for KNN classifier.

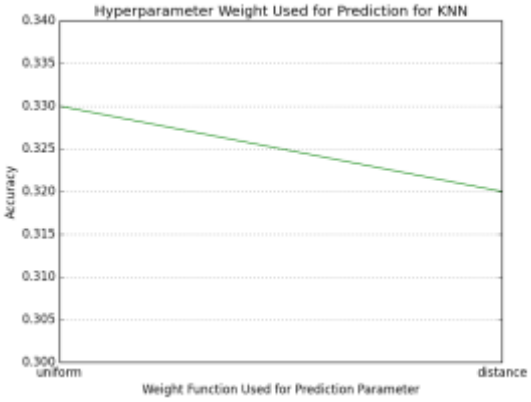

Figure 6. Weight function used for prediction vs Accuracy for KNN classifier.

The k-nearest neighbours algorithm classifies images based on the training examples in the feature space. Each image will be assigned the class of the majority of the k nearest neighbours. The KNN algorithm has an advantage over other algorithms in that it performs well with multi-modal classes given that it bases its decision on a neighbourhood of similar images [6]. One of the disadvantages that KNN has is that it weights all features equally, which may lead to classification errors [6]. The hyperparameters for the KNN trained were also adjusted by maintaining all except for one hyperparameter fixed. This was done for the number of neighbours, the weight function used for prediction, the algorithm used to compute the nearest neighbours, and the leaf size of the KNN. Only two of these hyperparameters showed variation when adjusting them. In Figure 5 we may observe the changes in the accuracy due to the variation in the number of neighbours considered for the majority vote. We can see that as the number of neighbours is incremented, the behaviour of the KNN is not stable, which might be an indicator of overfitting. Therefore, the default 5 neighbours is

selected, it is as high as the value obtained for 9 neighbours, but it is not in the threshold of the variations.

Figure 6 displays the variation of the accuracy with respect to the weight function used. The function may be uniform, in which all the points are weighted equally, or it can depend on distance, where the weight of the points given to a neighbour in the vote is greater as the distance is smaller. The uniform distance appeared to be better in this case, although the difference between one weight function and the other may not be statistically significant.

Figure 7 displays the accuracy of the methods studied over the public test set, (except for KNN whose accuracy is that of the validation set, due to the experimental limitation). At a first glance, it is evident that GIST features have an important role when classifying the image, offering more information than only the colour of a certain pixel when compared to the KNN results. The ResNet50 features classified with an SVM almost double the accuracy of the baseline. This might be due to the fact that these features are latent and offer more information to the computer than only colour or GIST. Now, comparing two models, ResNet50 + SVM + Augmented Data vs ResNet50+SVM + Error-Correcting, the latter model outperforms the former. This could be due to the fact that when training the SVM with error correction, the algorithm would use a binary code to represent each class when training, and it make the examples in the feature space project new points in the class space. It also appears that using data augmentation in this case accounts for overfitting, as the results over the public test set display an accuracy of 0.79, while the accuracy over the validation set is 93%.

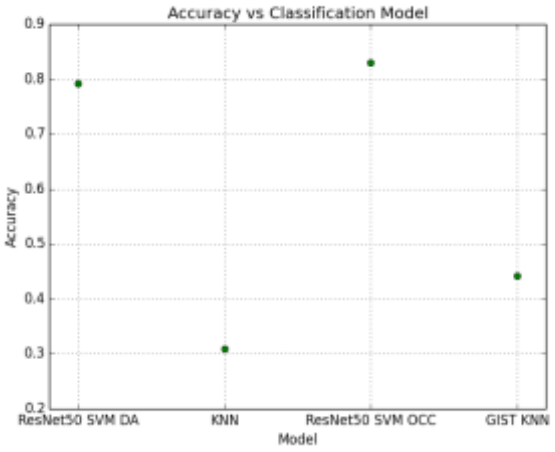

Figure 7. Accuracy of Various Models Over

IV. Conclusion

Transfer Learning for the problem of image classification is useful, specially using networks that use Residual Learning. This allows the network to learn the significant features with more flexibility. Additionally, it relieves the problem of the computational requirements of training a neural network from scratch. Using the features from the Maximum Pooling layer also reduces the amount of data that the classifier needs to handle. Instead of using the usual 128x128x3 (49152 data points per image), it uses the 2048 array that is outputted by the ResNet50 to perform the classification. Using Error Correction has proven useful in this problem, as it makes the classifier more robust. The Data augmentation is aiding in the accuracy; however, it is overfitting the data. Although the GIST features do increment the KNN's accuracy, this model is not performing at a high level as is the CNN. This is due to its simplicity, and that the features it needs to look at probably are not being generalized by the nearest neighbour approach.

Future work will include testing the augmented dataset with the error correction algorithm, and implementing ensemble methods to improve the accuracy. If there is a GPU available,

retraining the ResNet50 architecture with the provided 8 classes could also be beneficial for this problem.

V. References

[1] Torrey, L., Shavlik, J. Transfer Learning. University of Wisconsin. Url: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

[2] He, K. Zhang, X. Ren, S. and Sun, J. (2015) Deep Residual Learning for Image Recognition. Url: https://arxiv.org/pdf/1512.03385v1.pdf

[3] Chollet, F. "Fchollet/Deep-Learning-Models". GitHub. N.p., 2016. Web. 2 Dec. 2016.

[4] Nie, A., Ren, F., and Desai, N. (2016) RedFuse and ReFiNet:Enhanced CNN Architectures for Image Classifications.
Url: http://cs231n.stanford.edu/reports2016/277_Report.pdf

[5] Chollet, F. (2015). Keras. Github. Url:https://keras.io/getting-started/faq/

[6] Kim, J. Kim, B., and Savarese, S. (2012). Comparing Image Classification Methods: K-Nearest-Neighbor and Support-Vector-Machines. Applied Mathematics in Electrical and Computer Engineering, p133-138. ISBN:978-1-61804-064-0.

[7] 3.1. Cross-Validation: Evaluating Estimator Performance — Scikit-Learn 0.18.1 Documentation". Scikit-learn.org. N.p., 2016. Web. 2 Dec. 2016.

[8] Google,. "Tensorflow/Tensorflow". GitHub. N.p., 2016. Web. 2 Dec. 2016.

[9] Rosebrock, Adrian. "Imagenet Classification With Python And Keras - Pyimagesearch". PyImageSearch. N.p., 2016. Web. 2 Dec. 2016.

[10] Sklearn.Neighbors.Kneighborsclassifier — Scikit-Learn 0.18.1 Documentation". Scikit-learn.org. N.p., 2016. Web. 2 Dec. 2016.

[11] Sklearn.Svm.SVC — Scikit-Learn 0.18.1 Documentation". Scikit-learn.org. 2016. Web. 2 Dec. 2016.

[12] Fei-Fei, L. Karpathy, A. and Johnson, J. CS231 Lecture 11: CNNs in Practice. Stanford University.