FINAL REPORT ON SASSASSAS

SATELLITES AS SOURCES FOR SHIP AND SHORE SPOTTING AND SORTING

Luke Koch, Matt Lane, Jean Merlet, Alice Townsend
The University of Tennessee
Knoxville, TN 37996
lkoch2@vols.utk.edu, mlane42@vols.utk.edu, jmerlet@vols.utk.edu, atowns21@vols.utk.edu

December 9, 2020

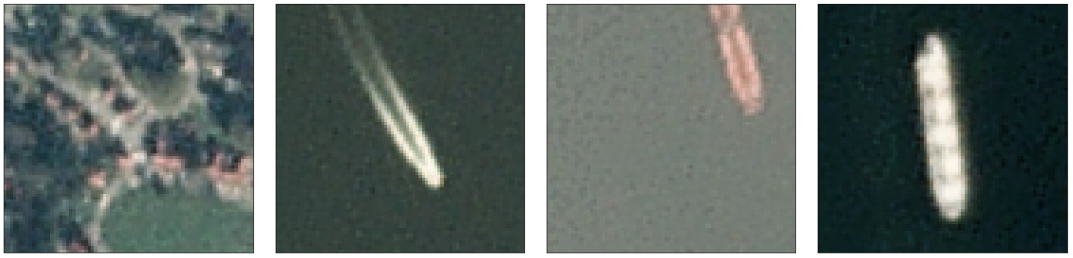
1 Introduction

1.1 Motivation

Ships utilize an Automatic Identification System (AIS) transponder which sends and receives information every few seconds to receiver devices of other ships or land systems. The AIS transponder was created with the goal to prevent collisions between ships at sea. The use of these transponders have become so prevalent that the data is not only utilized by the ships themselves but also by naval forces and the coast guard, port authorities, and other maritime authorities. A significant limitation of AIS is that it can only transmit its signal to approximately 50 nautical miles, which prevents a complete picture of maritime traffic on a much larger scale[1]. The Satellite Automatic Identification System (S-AIS) extends the range of AIS systems by sending and receiving signals to satellites, which provides a comprehensive picture of activity in areas that the maritime authorities are specifically interested in. S-AIS is very effective in tracking ships which are required to install the transponder, but cannot detect those which do not have a transponder installed or ships which turn off their transponders[2]. Satellites can help alleviate this problem by capturing images of the ships. However, the volume of images makes it difficult for various organizations to manually examine each image to detect the ships. Machine learning algorithms can solve this problem by creating automated systems that can analyze the images in real-time to detect the ships.

1.2 Dataset

The dataset consists of 4000 RGB images extracted from satellites over the San Pedro Bay and San Francisco Bay areas of California[3]. The 80x80 pixel images are orthorectified (to correct for curvature of the earth) and each pixel represents 3x3 meters. The following naming convention is used to describe each image: *label_scene id_longitude_latitude* where the *label* corresponds to the image's classification as either *ship* or *no-ship*. The data is stored in row-major order as a flat array of integers. The first 6400 entries are the red values, the next 6400 are the green values, and the least 6400 are the blue values. Of the 4000 images in the dataset, 1000 contain ships. The ship images are fairly consistent in certain regards. Specifically, the ships are always centered and tend to take up approximately the same amount of space in the image. However, the ship color, sea color, and ship orientation vary significantly. The remaining 3000 non-ship images, on the other hand, are highly inconsistent. They are divided into three categories: no ship at all (land, shore, or open sea), partial footage of a ship that does not meet criteria for classification as a ship, and commonly mislabelled images (strong linear features such as a bridge).



[Shore, Jet Ski, Partial Ship, Ship]

2 Initial Data Pipeline

Data obtained from Kaggle contained image and json data. Due to the ease of importing a singular json blob as opposed to iteratively adding sample data from each individual images, a json data loading module was written. Data could be loaded from the module in multiple ways:

- **load_data**: The base function simply retrieves the data from the json object via a user supplied file path as an input parameter. The function then reshapes the data into an array of 80×80 RGB values from one row per sample.
- **load_and_sample_data**: Retrieves the reshaped RGB data via the load_data and randomly samples from the data set, making use of the **randomlySample** function. The **randomlySample** function permutes the data, and returns a user specified number of samples (defaulted to the size of the entire set).
- **load_data_train_test_split**: In order to have a reasonable train/test split, the module has the function load_data_train_test_split. This function samples data randomly from the load_and_sample_data function, and splits 70 percent of the randomly sampled data into a training set, 15 percent into a testing set, and the remaining 15 percent into a validation set.

The module contains a **usage** function as well to assist users who may be unfamiliar with which particular loading function they would like to use.

3 Methods & Results

Project github: https://github.com/LaneMatthewJ/SaSSaSSaS

3.1 MPP with PCA and FLD

We trained an MPP Euclidean distance classifier on the original dataset to provide a baseline before examining the performance of our classifiers on a PCA-reduced version of the data. Our experimentation with different prior probabilities yielded Figure**??**. The [.75 .25] prior was based on the distribution of the training set. In lieu of domain expertise, this seemed like a reasonable approach. We compared the accuracy it yielded with an assumption of equal probabilities [.5 .5] and an assumption that ships are highly unlikely [.9 .9].

Algorithm	PCA	Priors	Overall Accuracy	Classwise Accuracy	Runtime (s)
Euclidean	N/A	.7425, .2575	.7500	1 , 0	14.4432
Euclidean	N/A	.5, .5	.6767	.6867 , .6467	14.9656
Euclidean	N/A	.9, .1	.7500	1 , 0	14.6476

We generated corresponding confusion matrices.

Algorithm	PCA	Priors	Confusion Matrix
Euclidean	N/A	.7425 , .2575	$\begin{Bmatrix} 450 & 0 \\ 150 & 0 \end{Bmatrix}$
Euclidean	N/A	.5 , .5	$\begin{Bmatrix} 309 & 141 \\ 53 & 97 \end{Bmatrix}$
Euclidean	N/A	.9 , .1	$\begin{Bmatrix} 450 & 0 \\ 150 & 0 \end{Bmatrix}$

Ergo, our best accuracy on the original data was .7500 with abysmal classwise accuracy. We only tested the Euclidean classifier because Mahalanobis and quadratic took far too long to train on the original data. At this point, we began dimension reduction. Our first reduction to 850 dimensions preserved 99.02% of the variation in the data and yielded the following with a Euclidean distance classifier:

Algorithm	PCA	Priors	Overall Accuracy	Classwise Accuracy	Runtime (s)
Euclidean	850	[.7425, .2575]	.7500	[1 , 0]	.0828
Mahalanobis	850	[.7425, .2575]	.9067	[.9333 , .8267]	28.6392
quadratic	850	[.7425, .2575]	.7500	[1 , 0]	38.7694

g

These are the corresponding confusion matrices for the above classifiers.

Algorithm	PCA	Priors	Confusion Matrix
Euclidean	850	.7425, .2575	$\begin{Bmatrix} 450 & 0 \\ 150 & 0 \end{Bmatrix}$
Mahalanobis	850	.7425, .2575	$\begin{Bmatrix} 420 & 30 \\ 26 & 124 \end{Bmatrix}$
quadratic	850	.7425, .2575	$\begin{Bmatrix} 450 & 0 \\ 150 & 0 \end{Bmatrix}$

Since there was no loss in overall accuracy, we continued the dimension reduction down to 245 dimensions. This preserved 94.98% of the variation and produced rapid results on all three classifiers.

Algorithm	PCA	Priors	Overall Accuracy	Classwise Accuracy	Runtime (s)
Euclidean	245	[.7425, .2575]	.7500	[1, 0]	
Mahalanobis	245	[.7425, .2575]	.9367	[.9511, .8000]	.8338
quadratic	245	[.7425, .2575]	.7850	[.7800, .8000]	1.3220

Confusion matrices for the above classifiers.

Algorithm	PCA	Priors	Confusion Matrix
Euclidean	245	.7425, .2575	$\begin{Bmatrix} 450 & 0 \\ 150 & 0 \end{Bmatrix}$
Mahalanobis	245	.7425, .2575	$\begin{Bmatrix} 428 & 22 \\ 30 & 120 \end{Bmatrix}$
quadratic	245	.7425, .2575	$\begin{Bmatrix} 351 & 99 \\ 30 & 120 \end{Bmatrix}$

Next we applied 5-fold cross validation.

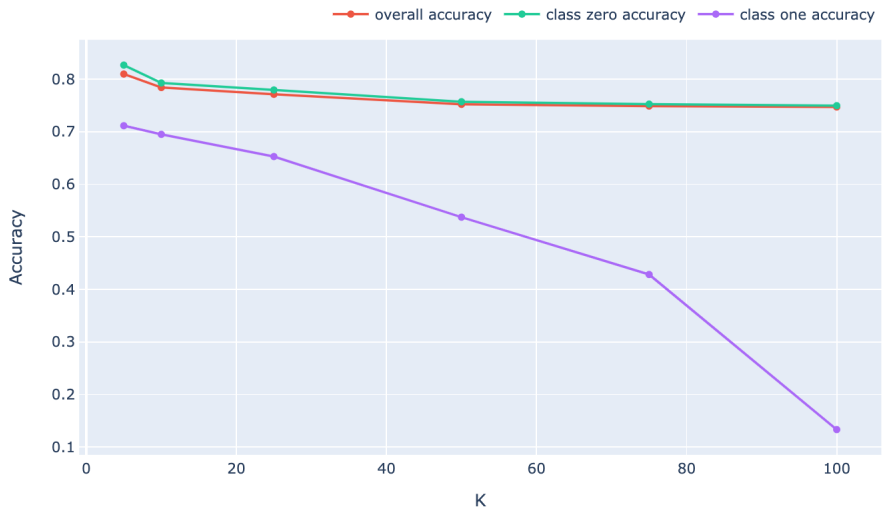
Algorithm	PCA	Priors	Average Overall Accuracy	Average Classwise Accuracy	Average Runtime (s)
Euclidean	245	[.7425, .2575]	.7518	[1, .0071]	.0195
Mahalanobis	245	[.7425, .2575]	.9224	[.9561, .8212]	.9089
quadratic	245	[.7425, .2575]	.7965	[.8165, .7365]	1.4124

We were pleasantly surprised by the Mahalanobis distance classifier's performance in conjunction with PCA (245 dimensions preserved). Although this performance is not state of the art, it is extremely lightweight and quick. This might make it useful on supervisory control and data acquisition (SCADA) devices or cyberphysical security systems with limited memory and processing power. We experimented with creating a boosted version of Mahalanobis in the hopes of creating an ultra-fast yet accurate classifier, but ran into difficulties in term of finding a productive way to weight the data in successive runs. Nonetheless, Mahalanobis performed better than expected. The classwise accuracy of this classifier [.97,.81] is also worth noting, as it is a dramatic improvement over Euclidean and quadratic. Since the performance of the quadratic classifier was between that of the Euclidean and Mahalanobis distance classifiers, we believe that an underlying assumption that the data is Gaussian is not harmful for his dataset. With regard to prior probabilities, we found that an assumption that the classes are evenly distributed lowered our accuracy. When we decreased the probability of spotting a ship below the 25% of our training data, we did not suffer a reduction in accuracy. Therefore, we believe there is a threshold with regard to the prior probability of spotting a ship. Once you assume a ship is rare, exactly how rare it is has little effect on accuracy for this data set.

We also tested Fisher's Linear Discriminant in conjunction with MPP classifiers. However, the performance was no better than random chance. This poor result makes intuitive sense. Since FLD reduces a dataset to $c - 1$ dimensions where c is the number of classes, FLD is attempting to reduce an 80x80 three-color image down to a single monochrome pixel. It hardly seems surprising that this sort of extreme reduction would obliterate any useful trace of a ship in the image. Perhaps if we were detecting an extremely large or bright flash in the image we would have had better performance.

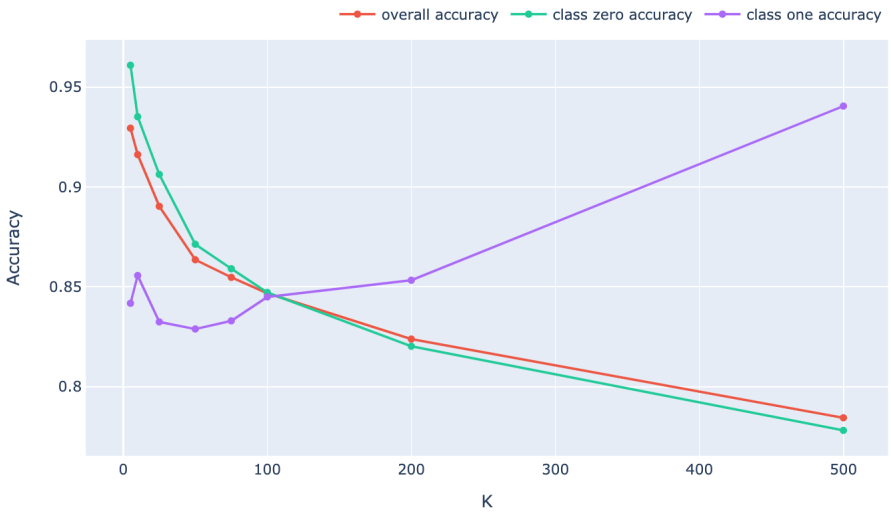
3.2 k-Nearest Neighbors

K-fold cross validation was used to train the k-nearest neighbors algorithm with both dimensionality reduction and without dimensionality reduction. The number of neighbors, k, was varied from five to one hundred when dimensionality reduction was not utilized. The figure below displays the overall accuracy and class-wise accuracies while varying the number of neighbors within the k-fold cross validation. The optimal choice for the number of neighbors was five.



The final model, with the number of neighbors selected as five, resulted in a class one accuracy of 0.75 and overall accuracy of 0.81 using the test dataset. The results of the final kNN model, with dimensionality reduction, are displayed in the table below. Principal component analysis (PCA) was then utilized before re-training the k-nearest neighbor algorithm to try and reduce noise within the dataset while preserving 99% of the original variability. This resulted in selecting 833 principal components. The optimal number of neighbors was equal to ten which was selected using the class-wise accuracies and overall accuracy from the k-fold cross validation. The selection of the number of neighbors was determined using the figure below. We can see that the overall accuracy and class one accuracy are maximized when the number of neighbors is selected to be ten.

Algorithm	PCA (error %)	Avg. Overall Accuracy	Avg. Classwise Accuracy	Runtime (s)
kNN (k = 5)	N/A	0.81	[0.82, 0.75]	61.09
kNN (k = 10)	0.01	0.92	[0.95, 0.83]	7.65



After reducing the dimensionality of the dataset, the overall accuracy increased to 0.92 and the class one accuracy increased to 0.83 utilizing the testing dataset. The results of the final kNN model, with dimensionality reduction, can be viewed in the table above.

3.3 Support Vector Machines

The support vector machine (SVM) with a linear, radial basis function, and third degree polynomial kernel were trained both without and with PCA. The best classifier, without dimensionality reduction, was the SVM with a radial basis function kernel. The regularization parameter, C, was tuned using cross validation which dictates the width of the margin. For the radial basis function kernel the optimal value of C was ten which resulted in an overall accuracy of 0.976 on the cross validation dataset and 0.98 for the overall accuracy on the test dataset. When applied to the testing datset this model retained 0.98 overall accuracy. The best classifier, with dimensionality reduction, was the SVM again with a radial basis function kernel. The optimal value for C was one hundred which obtained 0.97 on the overall accuracy for the cross validation set and 0.98 for the overall accuracy on the test dataset.

Algorithm	PCA (error %)	Avg. Overall Accuracy	Avg. Classwise Accuracy	Runtime (s)
SVM (RBF/ C = 10)	N/A	0.98	[0.98, 0.97]	60.99
SVM (RBF/ C = 100)	0.01	0.98	[0.98, 0.95]	2.62

3.4 Random Forest Classifiers

Random Forest Classification with warm start and 100 estimators were trained with all features, varying amounts of principal components, and K fold accuracy scoring. Additionally, for a greater accuracy, all random forest classifiers were trained with a warm start. When applying K-fold cross validation on the Random Forest Classifier, we found that the average over 10 folds was 0.947 accuracy. The random forest classifier achieved an accuracy of 0.965 on the train / test split, however, this highest accuracy was achieved by using only 50 principal components, with a percentage explained of 0.85. As can be seen with the BPNN classifier as well, a current hypothesis as to why the accuracy improves with a removal of principal components is that the principal components removed essentially denoise the data. We can see this trend almost entirely in the class 1 accuracy increase as we remove components from 850 to 50.

Algorithm	PCA	Avg. Overall Accuracy	Avg. Classwise Accuracy	Avg. Runtime (s)
Random Forest	N/A	.96	[.98, .90]	26.745
Random Forest	850	.8933	[1.0, .5733]	6.025
Random Forest	500	.9183	[1.0, .673]	4.3906
Random Forest	250	.943	[.1.0, .73]	3.722
Random Forest	100	.9583	[.995, .82]	2.323
Random Forest	50	.965	[.995, .85]	2.074

3.5 Back Propagation Neural Network Classifiers

Given that Neural networks can vary significantly in their architectures, yielding wildly different results, a number of architectures were tested, varying the hidden nodes (HN), solvers, and application of dimensionality reduction. The two methods of solvers used were Stochastic Gradient Descent (SGD) and L-BFGS, an approximation of the Broyden-Fletcher-Goldfarb-Shanno algorithm, which estimates an inverse Hessian matrix for its search of optimal weights. When applying K fold cross validation to our BPNN of highest accuracy (i.e. BPNN L-BFGS with 1000 Hidden Nodes with 250 principal components), we found it had an average accuracy of 0.956. It does raise the question of accuracy increasing with the reduction of dimensionality and ultimately in increase in overall error within the dataset. Our current hypothesis remains as above in the random forest section that the reduction of features acts as a denoising agent within this dataset.

Algorithm	PCA	Avg. Overall Accuracy	Avg. Classwise Accuracy	Avg. Runtime (s)
BPNN SGD 100HN	N/A	.75	[1.0, 0.0]	34.73
BPNN SGD 1000HN	N/A	.75	[1.0, 0.0]	1517.64
BPNN SGD 1000x1000HN	N/A	.75	[.1.0, 0.0]	318.503
BPNN L-BFGS 100HN	N/A	.9517	[.9644, .9133]	698.933
BPNN L-BFGS 1000HN	N/A	.95	[.9666, .9]	4586.674
BPNN L-BFGS 1000x1000HN	N/A	.75	[1.0, 0.0]	613.756
BPNN SGD 1000HN	850	.9316	[.96, 0.8466]	17.365
BPNN SGD 1000HN	500	.9483	[.9688, 0.8867]	8.24
BPNN SGD 1000HN	250	.9533	[.96, 0.9333]	12.428
BPNN SGD 1000HN	100	.9316	[.977, 0.7933]	2.77
BPNN SGD 1000HN	50	.961	[.9655, 0.92]	4.231
BPNN L-BFGS 1000HN	500	.9633	[.9688, 0.9466]	26.757
BPNN L-BFGS 1000HN	250	.965	[.9711, 0.9466]	18.731
BPNN L-BFGS 1000HN	100	.975	[.9866, 0.94]	12.625
BPNN L-BFGS 1000HN	50	.9683	[.9755, 0.9466]	11.947

3.6 K-means and WTA

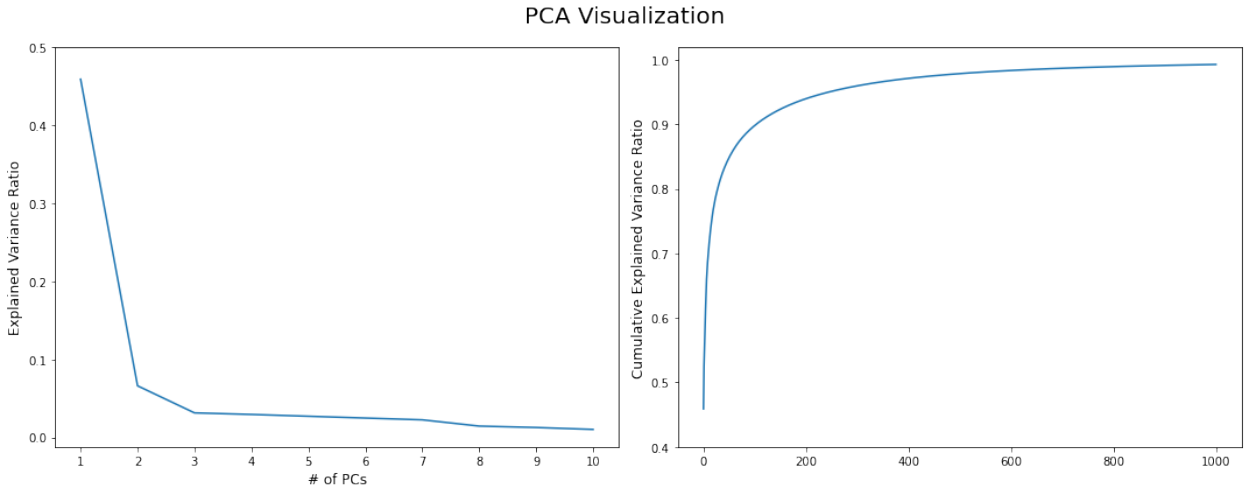
For clustering methods, we implemented both K-means and WTA. While WTA slightly outshone K-means in both accuracy and runtime, both algorithms faired extremely poorly. The accuracies were calculated with 5-fold cross validation. We suspect that these very low accuracies are due to the extreme variability across images of subsections of the images. For the ship pictures, there is almost no consistency except for the outermost edges, which are very often sea, or for the exact center, which is often a ship - but even these parts of the image vary in color. The non-ship images vary much more wildly than the ship images, so it is not too surprising that the clustering results based on subsections of the images was barely better than random chance.

Algorithm	PCA	Avg. Overall Accuracy	Avg. Classwise Accuracy	Avg. Runtime (s)
K-means	N/A	.5353	[.6926, .3922]	6.99
K-means	245	.5352	[.6933, .3916]	2.28
K-means	2	.5362	[.5920, .4932]	.0195
WTA	N/A	.5265	[.3674, .6677]	8.06
WTA	245	.5238	[.3789, .6868]	.94
WTA	2	.5265	[.3791, .6864]	.46

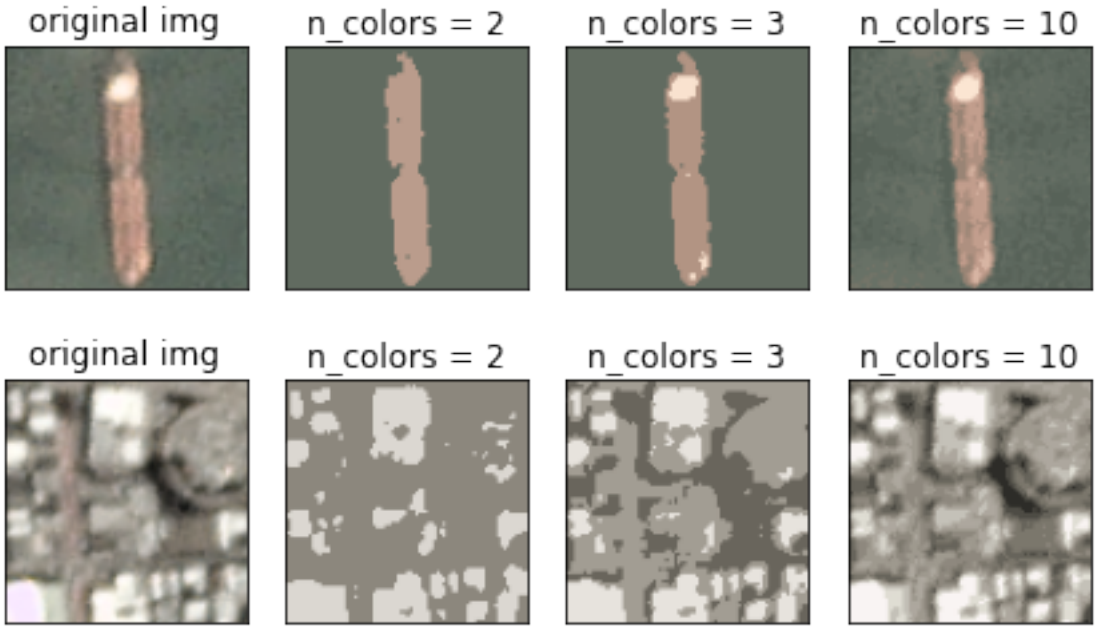
3.7 Dimensionality Reduction and Denoising with K-means

As explained in the MPP section, FLD doesn't make any sense for a binary image classification goal. However, all of the classifiers faired well with different levels of PCA. To attempt to understand why certain numbers of principal components were chosen, we can visualize the explained variance ratios at different numbers of PCs. While the first principal component represents vastly more of the variance (0.45) than any of the other components, with the second at 0.8, it takes until over 800 principal components to achieve 0.99 explained variance. Thus it makes sense that there is not an enormous increase in accuracy between 200 and 800 principal components for MPP and also makes sense that certain classifiers such as Random Forest perform better on a lower number of PCs. For Random Forest, a few hundred

principal components covering only a few percentage points of variance merely represent noise, while for MPP they represent valuable information with which to eke out a few more percentage points of accuracy.



Feeling bad for K-means, we decided to see if might be useful in any other way. Specifically, we implemented a color reduction scheme using K-means to see if this might serve to denoise the images in the dataset without removing potentially important sources of variance as is the case when using too few principal components in PCA. At first, clustering each image in a 4000 image dataset sounds like it may be highly computationally intensive. However, there is not that high a number of different pixels in any image, let alone an 80x80 image such as we have in our dataset. Therefore, it was more than sufficient to randomly sample 1000 of the pixels from the 6400 in each image to cluster on, and could possibly have been done with even fewer pixels. The result was that reducing all of the images to 2 colors took less than 2 minutes, and even reduction to 100 colors could be done in a few minutes. Below are examples at 2, 3, and 10 colors for a ship and no-ship sample, respectively.



It is clear that visually, as few as 2 or 3 colors appear to capture almost the entirety of the important color variation in the images to the human eye for the simpler picture of the ship. However, even in the more complex on-shore image, as few as 10 colors appears nearly identical to the original full-color image. Interestingly, the noisy color variation of the sea is smoothed to a single color when using 2 colors and containing a stark contrast such as a ship. Thus we

hypothesized that a very low number of colors in the 2 to 10 range may actually help certain classifiers. When testing out this hypothesis, the results indicated that more than 10 colors did not reduce classifier accuracy. However, while there was no loss in accuracy with 10 colors, there was no gain either, and using fewer colors down to 2 resulted in slight accuracy losses, suggesting that a slightly more complex color palette than 2 colors was necessary for the highest classifier performance.

This color reduction could be used as a critical step in a much more developed and complex pipeline. Specifically, sea-land separation is typically a critical early step in ship recognition for both accurate detection of ships in harbors and to negate the high rate of false alarms created by on-shore objects[4], typically achieved by incorporation of a land-sea map to the dataset. Clearly, a land-sea map is not quite as important for this dataset, as people are obtaining 0.99 accuracy without one by implementing a CNN. However, it could prove useful for a less forgiving ship classification task such as when ships are not nicely pre-centered and cropped in the satellite image.

3.8 Fusion

Having seen how both all of the classifiers we had learned over the semester and the SOTA CNN performed, we applied Naive Baysian Fusion to each of our favorite classifiers from our individual classification attempts. As there are four of us, this resulted in Mahalanobis MPP, RBF SVM, BPNN, and K-means as the four classifiers we fused. We fused 3 of the 4 in each of the 4 combinations, but runtime was of course nearly instantaneous and is thus not worth listing.

Combination	Overall Accuracy	Classwise Accuracy
MPP, SVM, BPNN	.9733	[0.9865, 0.9351]
MPP, SVM, K-means	.9617	[0.9863, 0.8944]
MPP, BPNN, K-means	.9283	[0.9700, 0.8204]
SVM, BPNN, K-means	.9600	[0.9863, 0.8889]

It is not particularly surprising that the winning combination is MPP, SVM, and BPNN because K-means performed so abysmally badly. Nevertheless, the three combinations that did include K-means did not have horrendous accuracy, which suggests that Baysian Fusion is able to largely counter the effects of horrendously performing classifiers as long as they are in the minority. We confirmed this suspicion by Fusing various well-performing classifiers with K-means and WTA, which resulted in quite poor accuracy. Also of interest is that the fused results of the best combination do not outperform all of the individual results. Namely, the MPP, SVM, and BPNN combination slightly outperforms MPP and BPNN but underperforms compared to SVM.

4 State of the Art

In general, convolutional neural networks represent state-of-the-art performance on satellite image classification tasks. With regard to academic papers concerning this data set, Nugroho and Windarto published the best results[5]. They compare and contrast two slightly different Convolutional Neural Networks (CNNs) with an unsupervised method for image classification known as Grey Level Co-occurence Matrix (GLCM). The two CNNs differ in their final max pooling layer. The first CNN retains location information by pooling into a 3x3x*n* layer where *n* is the number of channels. The second flattens the data into a 1x1xn layer. The paper demonstrates that the location-aware CNN strongly outperforms the other two methods. It should be noted that the CNN architecture employed in this paper is a depth-wise convolution that works well with multi-channel data (RGB channels). Interestingly, this depthwise approach performs extremely well on a greyscale version of the images as well.

Feature from	Dataset			
	Ship		EuroSAT	
	Gray	RGB	Gray	RGB
ConvNet-1	0.9640	0.9708	0.7162	0.786
ConvNet-2	0.9190	0.9258	0.7124	0.7268
GLCM	0.8950	0.9250	0.5014	0.6318

Figure 1: Nugroho Accuracy

	Class	Precision	Recall	F1-score
ConvNet1	0	0.98	0.98	0.98
	1	0.94	0.94	0.94
ConvNet2	0	0.96	0.94	0.95
	1	0.83	0.87	0.85
GLCM	0	0.95	0.95	0.95
	1	0.85	0.84	0.84

Figure 2: Nugroho Performance Metrics

The overall architecture consists of 4 convolution blocks followed by the adaptive pooling and flattening layers (where the two CNNs differ) and a final fully-connected layer.

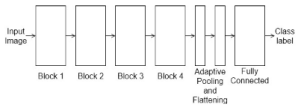


Figure 3: Nugroho CNN Architecture

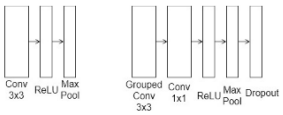


Figure 4: Nugroho Block Architecture

With regard to Kaggle notebooks, we found it difficult to compare results when each notebook used a different train/test split and reported only test or validation accuracy rather than both. In order to create a fair comparison, we implemented one of the notebooks[6] we found and trained it using our train/test split. We then validated it using the same stratified K-fold cross validation method as our other classifiers.

Algorithm	Average Overall Accuracy	Average Classwise Accuracy	Average Runtime
CNN	.9918	[.9937, .9859]	71.83

5 Discussion

Interestingly, almost all of the Kaggle notebooks focused on attempts at classification with CNNs. While CNNs are SOTA for image classification, we were able to achieve nearly identical accuracies with other, arguably much easier-to-implement algorithms. Namely, SVM with an RBF Kernel, Random Forest after a bit of parameter searching, and MPP with Mahalanobis Distance as the metric all performed in the upper 90s of accuracy. Most striking is perhaps the result of the Mahalanobis classifier, which scored 0.9367 accuracy on the first 850 principal components after applying PCA, but with a runtime of under a second. Nevertheless, the best Kaggle CNNs did outperform our best non-CNN classifiers, but not by much, and there were plenty that we beat. Indeed, many of the Kaggle notebooks only scored in the upper 80s or low 90s of accuracy, which means they were beaten by our 850 PC Mahalanobis classifier. It's quite possible that with further hyperparameter tweaking, fusions of our classifiers, and various other dimensionality / denoising efforts we may be able to equate or beat the CNN results.

Another important consideration is that the majority of our classifiers tended to perform better classifying class 0 (no-ship) than class 1. This is most likely due to the fact that there are 3 times as many non-ship images as there are ship images. However, our fusion results might be greatly increased if we could find a classifier or tweak one of our current classifiers to return highly accurate results that also perform best in the class 1 (ship) category, thus allowing us to fuse the strengths of different classifiers.

6 Code README

Rather than submit a pdf with long sections of code mixed in, we opted to compile our results in this document and provide a reference here to the code in the accompanying zip file.

File Name	File type	Contents	Author
SVM & KNN.ipynb	Jupyter Notebook	kNN, SVM	Alice Townsend
clustering_methods.ipynb	Jupyter Notebook	K-means, WTA, PCA plots	Jean Merlet
fusion.ipynb	Jupyter Notebook	Naive Baysian Fusion	Jean Merlet
randomForestandBPNN.ipynb	Jupyter Notebook	Random Forest, BPNN	Matt Lane
mpp.ipynb	Jupyter Notebook	MPP+PCA	Luke Koch
fld.ipynb	Jupyter Notebook	MPP+FLD	Luke Koch
sota_cnn.ipynb	Jupyter Notebook	State of the Art CNN	Luke Koch and T. Rasymas
mpp_classifiers.py	Python Source Code	MPP classifiers	Dr. Hairong Qi
load_ship_data.py	Python Source Code	Data Loader	Matt Lane
load_flat_ncolor_data.py	Python Source Code	N-color data loader	Jean Merlet

References

[1] H. Ball, *Satellite AIS*. 2013.

[2] J. Gambrell, "Iran oil tanker pursued by us turns off tracker near syria," *AP News*, 2019. https://apnews.com/article/4f3f11c2f789403b8945087dc2c39313.

[3] Planet Team, San Francisco, CA, *Planet Application Program Interface: In Space for Life on Earth*, 2017. https://api.planet.com.

[4] U. Kanjir, H. Greidanus, and K. Oštir, "Vessel detection and classification from spaceborne optical images: A literature survey," *Remote Sensing of Environment*, vol. 207, pp. 1–26, Mar. 2018.

[5] K. A. Nugroho and Y. E. Windarto, "Analyzing depthwise convolution based neural network: Study case in ship detection and land cover classification," *Jurnal Ilmu Komputer dan Informasi*, Jul 2019.

[6] T. Rasymas, "Cnn classifier using keras," 2017. https://www.kaggle.com/tomras/cnn-classifier-using-keras.