Classifying the "largest" digit

Comp551-002, Winter 2018

Yann Foucault
yann.foucault@mail.mcgill.ca
260298587

Philippe Min
philippe.min@mail.mcgill.ca
260637157

Mehrzad Mortazavi
mehrzad.mortazavi@mail.mcgill.ca
260821564

Kaggle Team Name: !free_lunch!

***Abstract*—Faced with a dataset of complex images, image preprocessing is key to improving the predictive performance of any learner. Support Vector Machine ("SVM") learners and Feedforward neural networks ("FNN") struggle to reach high accuracy. Alternative learners, in particular convolutional neural networks, perform much better.**

Keywords—image, preprocessing, learner, neural networks, convolutional neural networks

I. Introduction

MNIST dataset[1] is a state-of-the-art set of images of handwritten digits, from which feedforward neural networks ("FNN") can learn reasonably well how to correctly identify a digit that has been written by hand. Even with minimal preprocessing,[2] Nielsen's FNN code gets a more than 80% accuracy rate on the MNIST test set.[3] The dataset we are confronted with is more complex, as explained in Section II.A. Hence the need to go through additional steps and explore alternative learners in order to get a decent accuracy rate when predicting labels for a previously unseen dataset.

II. Feature Design

Data preprocessing is one of the important steps of developing a Machine Learning pipeline, since it has a direct impact on the performance of the learning algorithm. We have used several Image Processing and Computer Vision techniques at this step. Our feature design and selection can be described as follows.

A. Exploring the Dataset

The dataset is composed of 50,000 training and 10,000 test images. It is based on the famous MNIST hand-written digits' dataset, already mentioned; some digits were selected randomly, rotated, magnified or reduced in size, and scattered on a different range of background images.

B. Removing the background and centering the digits

To remove the background, multiple thresholding algorithms were used, including Otsu thresholding, Adaptive thresholding, and binary thresholding; we realized that the best approach is to extract pixels with a value equal to 255 since the digits are pure white with this pixel value.

C. Finding the largest digits

First, we calculated the edge around each digit, then a bounding box around these contours was calculated to extract the largest digit.

D. Centering the image

We used segmentation methods to find the centroid of each component and by this information, we transferred the largest digit to the center of the image; this was done to remove potential correlation between the position of the digit on the image and its corresponding class.

E. Reducing image size

The original dataset takes 1.6 GB of memory after loading the csv file; each image has 4,096 feature points which after this extraction of the largest digit and centralization, most of the pixels are zero and some others are 255. Consequently, it takes more time and computational effort to train and test a model on this dataset.

After centering the largest component in image, we used a 32 by 32 pixels window of the preprocessed image, and by using int8 datatype instead of the default float64, our final processed image took only 50MB of RAM and reduced the training time from multiple hours to minutes.

Figure 1- Steps of preprocessing

[1] A database developed under Yann LeCun, from New York University, available at http://yann.lecun.com/exdb/mnist/.
[2] Minimal, i.e. not involving image transformation.
[3] Michael A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

III. Algorithms

A. SVM, a baseline linear learner

SVM is a linear classifier, meaning that it classifies data by separating them in the feature space using a straight line, or, in higher dimensions than 2, a hyperplane. (assume there are only two classes for now). Now, there is an infinite number of hyperplanes that can separate the same two subsets in a dataset. SVM allows us to identify the hyperplane that separates the two subsets with the largest margin. A key limitation is that, if the data is not linearly separable, SVM is in advance bound to fail at properly classifying data.

B. Feedforward Neural Network

A feedforward neural network ("FNN") is a network made of three layers or groups of layers: an input layer, fed by the feature vector associated with one image (a 4096-by-1 vector of integers ranging from 0 to 255 in our case), one or several hidden layers, and an output layer. There is no loop at any node or between layers, hence the term "feedforward". A fully connected FNN is one where every node in a given layer is linked to every node in the next layer. According to the model, at the output layer stage, the higher the value returned by the activation function from a given node, the more probable the target value associated with that node.

A key limitation of FNN when applied to images is that the algorithm cannot "see" any pattern or shape in the image, only the number of pixels and their respective values, which could just as well be scattered randomly on the frame.

C. K-Nearest-Neighbor

K-Nearest-Neighbor ("K-NN") algorithm is, unlike the other learners described in this section, a non-parametric learning algorithm that memorizes the training set and predicts target value of a new datapoint by using the K-nearest datapoints.

D. Decision Tree

A decision tree is an algorithm that has the shape of a tree, hence its name. Going from larger branches to smaller branches to leaves, one can decide at each node how to classify a given input by considering a specific feature against a threshold or criterion. Decision trees thus allow a richer partitioning of the input space.

E. Kernalized SVM

A kernalized SVM is an SVM where a kernel, i.e. a function that performs a dot product for some mapping φ of the input features. Using the dual form of the SVM, the so-called "kernel trick" allows to solve for the Lagrange multipliers for the constraints without even computing the feature mapping φ.

F. Ensemble learning algorithms

Ensemble learning is an approach to take the advantage of multiple learning algorithms by combining their predictions to improve the overall performance and reduce the impact of bad assumptions that each learner might have. In the following, we have covered some methodologies that we have implemented: 1. Random Forest uses K replication of trainingset (Bootstrap) created by sampling with replacement to train K different Decision Trees and utilizes information gain to choose the best learner; 2) Bagging uses Bootstrap datasets to train multiple classifiers and averages their results as the final prediction of the model; 3) Boosting: in Boosting, a learning algorithm is sequentially trained and a weighted committee vote of these algorithms will decide the final prediction. In each step, weights assigned to each datapoint is modified to force the learner to minimize the error.

G. Convolutional Neural Networks

Convolutional neural networks ("CNN") are deep, or multi-layer, neural networks where layers applying a convolution in the mathematical sense, hence the name, alternate with pooling and fully connected layers. Key features are that the algorithm can focus on some data features that only depend on a region of the image (local receptive fields); parameters are shared, using a filter; and pooling or subsampling is performed on larger and larger areas as we go deeper into the network, so that key features, whatever their location in the image, can be taken into account. CNN have achieved great feats in the field of image recognition these past few years.

IV. Methodology

To get the most out of each learners and help identify the best of them, we fine-tuned the hyperparameters using cross-validation. For FNN, as well as for CNN, the most promising learner based on cross-validation results, we tested their accuracy on the Kaggle test dataset.

A. Linear SVM

We used GridSearch and 5-fold cross validation to find the best linear SVM model using varations of hyperparameters.

B. FNN

We encoded the output digit into a one-hot vector of size 10-by-1, so that the digit predicted by FNN is the index of the vector element with the value closest to 1 (i.e. the highest probability of being true).

We applied a 80/20 ratio when splitting the dataset between training dataset and validation dataset.

When fine-tuning hyperparameters, we performed a grid search and varied some key hyperparameters such as learning rate, number of nodes in hidden layer, batch size, and weight and bias value initializing. We selected the number of hyperparameters that yielded the highest

average accuracy rate when taking the average of the accuracy rates at epoch 30 and epoch 100.

We varied the size of the mini-batch as part of hyperparameter fine-tuning. We also compared initializing the values of the weights and biases by generating them randomly from a standard Gaussian distribution (with mean 0 and standard deviation equal to 1) and from a uniform distribution between 0 an 1.

To train the network, we applied a mini-batch stochastic gradient descent to the FNN. We did not apply any improvement, such as momentum or adaptive learning rates, to the optimizer.

We ended up with a combination of hyperparameters that showed relatively honest results for a FNN applied to complex images, including: learning horizon set to 100 epochs; quadratic loss function; sigmoid as activation function; a single hidden layer. And we focused our energy on more promising models.

C. K-Nearest-Neighbor

Since k-NN makes decisions based on the values of the neighboring datapoints, considering the "neighbor" (choosing a good *k*) and access to good neighbors (enough generalization in the data points), has a direct impact on the performance.

D. Decision Tree

Decision tree is a powerful learner and we should restrict its capability in order not to overfit. First we trained a decision tree without any restriction to have a sense about its maximum depth. Then we trained multiple models with variations of maximum depth, criterion, and minimum samples split.

E. Random Forest

Since random forests are based on decision trees, we tuned the same hyperparameters in our experiments. We used Grid Search and 5-fold cross validation to find the best combination of hyperparameters.

F. Bagging

As an ensemble learning algorithm, Bagging is a computationally intensive method. We used k-NN as the estimator and tried a different number of estimators using 5-fold cross-validation and GrdiSearch.

Model #	Pre-processing	Mini-batch size	Nodes in hidden layer	Learning rate	Parameter initialization	Accuracy rate at step 30	Accuracy rate at step 100	Average
1	No	10/30/100	30/200	0.1/1/10	Gaussian	10%	10%	10%
2	Yes	30	30	0.1	Gaussian	26%	32%	29%
3	Yes	30	30	0.3	Gaussian	23%	44%	34%
4	Yes	30	30	1	Gaussian	22%	37%	29%
5	Yes	30	10	0.3	Gaussian	16%	30%	23%
6	Yes	30	100	0.3	Gaussian	24%	23%	24%
7	Yes	30	100	0.1	Gaussian	30%	36%	33%
8	Yes	30	300	0.1	Gaussian	31%	37%	34%
9	Yes	30	300	0.3	Gaussian	34%	43%	39%
10	Yes	30	300	1	Gaussian	34%	43%	38%
11	Yes	30	1000	1	Gaussian	38%	37%	37%
12	**Yes**	**100**	**1000**	**1**	**Gaussian**	**42%**	**54%**	**48%**
13	Yes	300	1000	1	Gaussian	36%	39%	37%
14	Yes	100	1000	1	uniform	11%	10%	11%
15	Yes	100	1000	1	uniform	10%	10%	10%

TABLE I. HYPERPARAMETER FINE-TUNING FOR FNN

G. Boosting

The number of estimators used for committee voting has direct impact on the performance of Boosting ensemble learning algorithm. We used k-NN as the estimator since it has already proved its performance in our experiments and it was possible to train it with our limited resources.

H. Kernalized SVM

Linear SVM is limited to linearly separable decision boundary problems. By combining SVM and kernalizing methods, we can transfer a non-linear decision boundary to a linearly separable problem and then use SVM to learn the model. We used multiple kernel functions including rbf, sigmoid, different orders of a polynomial in 5 fold cross-validation and GridSearch to find the best model.

I. Convolutional Neural Networks (CNN)

Similarly to FNN, we have split the data in a 80/20 ratio to create the training and validation datasets.

We are using an Adam optimizer, alongside a cross-entropy loss function for our model. Cross-entropy loss measures the disparity between the predicted probabilities of the output against the real label. It is a standard loss function used in multi-classification tasks in neural networks. The Adaptive Moment Estimation (Adam) optimizer is an adaptive learning rate algorithm that leverages the first and second moments of the gradients. Combined with cross-entropy loss, it is an effective optimization algorithm for deep learning models on noisy problems.

We have also included batch regularization and dropout to our models. Batch regularization stabilizes the gradient estimates in deep networks, leading to faster training time. Dropout forces neurons to work with random neighbours by deactivating a subset of nodes in a layer. Batch regularization and dropout help combat possible overfitting that may occur during training.

We have performed grid search when fine-tuning key hyper-parameters such as learning rate, number of epoch, number of layers, number of nodes in each hidden layer, etc. We have experimented with a number of models with different architectures. We started off with a very simple model with 2 layers as a baseline. We tried to improve it by changing the parameters of the convolution layer, by adding new layers, etc. We chose the most promising resulting model based on accuracy on validation set, and kept generating new model iterations by building upon it. We have also selected the best learning rate, number of epochs, and batch size mainly based on model accuracy. We also considered possible symptoms of over/under fitting, and high/low learning rates by analyzing the model accuracy and loss over epoch graphs.

V. Results

In this section, we cover the results of the different learning algorithms. In all cases, we have used cross-validation and validation-set error to find the best model. Then, for FNN and for the most promising learner, CNN, based on validation set results, we tested its predictive accuracy on the Kaggle website dataset. We got approximately the same test error as our validation error for CNN. Our approach for CNN has good generalization.

A. Linear SVM

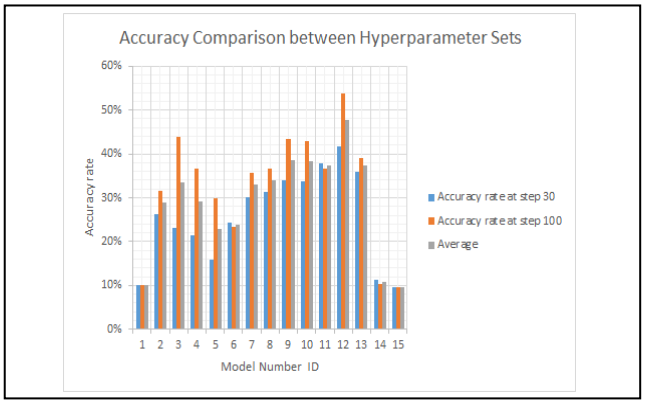
This learning algorithm assumes that data is linearly separable. This results in poor performance in comparison with other learning algorithms. We used GridSearch and 5-fold coross validation to find the best model which could predict the labels of the validation set with 73% accuracy.

B. Neural Network

For FNN, we began by training the network without preprocessing. Results were stubbornly poor, with an accuracy rate no better than random, at around 10%, regardless of the mini-batch size, the number of nodes in the network single hidden layer, and the learning rate (see

Fig. 1. Accuracy of various Hyperparameter Sets (using cross-validation)

Model 1 in Table I and Figure 1). Even normalizing inputs did not help, and gradient kept vanishing. When applied to preprocessed images (Models 2 to 15 in Table and Graph below), accuracy shot higher, to 54% on the validation set at the 100 epoch when initializing weights and biases



randomly from a standard Gaussian distribution, using a 100 mini-batch size, 1000 hidden layers, and a learning rate of 1 (Model 12 in Table I and Figure 1). On the test set on Kaggle website, such a combination of hyperparameters got 42.5% accuracy. Note how parameter initializing is crucial: when drawing parameters randomly from a uniform (0,1) distribution, the accuracy rate fell back to about 10%, so no better than without preprocessing (Models 14 and 15).

C. K-Nearest-Neighbor

Since the main hyperparameter in k-NN is choosing the *k* neighbors, we tuned *k* using 5-fold cross validation implemented in GridSearch.

K-NN performed pretty well on cross-validation set, reaching 82.3% accuracy, as can be seen in Table II. However, as can be expected from a lazy algorithm, k-NN has been a slow learner, taking more than 4 hours to train.

TABLE II. K-NN HYPERPARAMETER TUNING USING GRIDSEARCH

#	data points	time	n_neighbors	weights	Best parameters	Best candidate reslt
1	3000	1.2min	[5,10,15,20]	['uniform','distance']	'n_neighbors': 5, 'weights': 'distance'	82.3
2	50000	262.7min	[5,7,9]	['uniform','distance']	'n_neighbors': 5, 'weights': 'distance'	82.3

D. Decision Tree

Using GridSearch and 5-fold cross-validation, we were able to get 74.5% accuracy from decision tree (see Table III). As a starting point, we first trained a decision tree with default parameters implemented in Scikit tool, and it had 69% accuracy. But, after some fine-tuning, we were able to enhance the result We were able to enhance the result by 5%. Also, decision tree had a very short training time, when compared to lazy algorithms such as k-NN or kernelized SVM (see Table VII further below).

TABLE III. DECISION TREE HYPERPARAMETER TUNING USING GRIDSEARCH

#	data points	time	Max depth	criterion	Min samples split	Best parameters	Best result
1	5000	11.5s	range (4,60,4)	gini','entropy'	[2,4,8,16]	12, entropy, 2	72.7
2	50000	17m	range (4,60,4)	gini','entropy'	[2,4,8,16]	12, entropy, 2	74.5

E. Random Forest

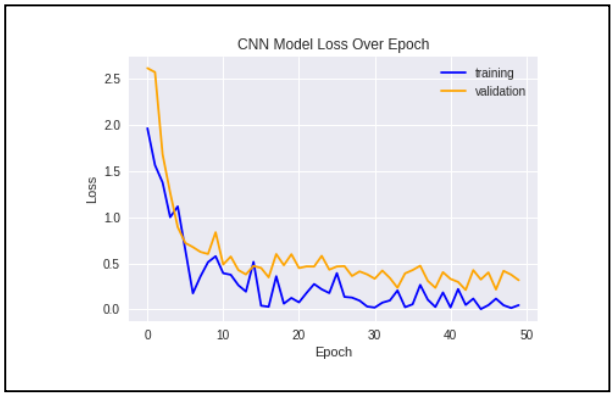
We were able to enhance the performance of decision trees by a great margin (87.3% vs. 74.5% accuracy on 50,000 datapoints, as can be seen in Tables IV and III respectively) by using random forests, which are multiple decision trees trained on bootstrapped datasets. In addition to the accuracy improvement of almost 13%, the random forest algorithm is even faster (14.3 min vs. 17 min. as shown in Tables IV and III respectively). We tuned the model by looking for the best combination of maximum depth, maximum number of features, criterion, and minimum sample splits.

TABLE IV. RANDOM FORSEST HYPERPARAMETER TUNING USING GRIDSEARCH

#	data points	time	Max depth range	Max features	criterion	Min samples split	Best parameters	Best result

1	50000	4.9m	(4,60,4)	default	['gini', 'entropy']	[2,4,8,16]	criterion='entropy', max_depth=44 , min_samples_split =8	87.3
2	5000	3.2m	(4,60,4)	(1,100,10)	['gini', 'entropy']	[2,4,8,16]	criterion='entropy', max_depth=24 , min_samples_split =8, max_features=60	78.8
3	50000	14.3m	(4,60,4)	(20,80,10)	['gini', 'entropy']	[2,4,8,16]	criterion='entropy', max_depth=24 , min_samples_split =8, max_features=81	87.5

Fig. 2. CNN Model Loss Over Epochs.



F. Bagging

Using this ensemble learning method, we were able to get great performance from training multiple kNN classifers. The main hyperparameter for Bagging is number of estimators that predict the target value. Using 100 estimators instead of 10, could enhance the result by 5%. The issue with these learning algorithm is they are computationally intensive and it was hard to train many models, due to time constraints.

TABLE V. BAGGING HYPERPARAMETER TUNING USING GRIDSEARCH

#	data points	Learner	estimators	time	Best parameters	Best result
1	10000	kNN	10	16.0min	n_neighbors=5, weights='uniform'	86.2
2	50000	kNN	10	449.2min	n_neighbors=5, weights='uniform'	90
3	5000	kNN	100	40.1min	n_neighbors=5, weights='uniform'	84.46

G. Boosting

As can be seen in TABLE VII, we couldn't get good results from Boosting even by using 100 estimators. Even though it is an ensemble learning algorithm, Boosting could not give good result on the validation and test set. We used kNN as the learning algorithm to be used for kNN.

TABLE VI. BOOSTING HYPERPARAMETER TUNING USING GRIDSEARCH

#	data points	Learner	estimators	time	Best parameters	Best candidate reslt
1	10000	kNN	10	1.8s	n_neighbors=5, weights='uniform'	45
2	50000	kNN	100	17	n_neighbors=5, weights='uniform'	61.4

H. Kernalized SVM

We got our best result with 3rd order polynomial kernel function (shown in Table VII). Unfortunately, we could not train the model using the whole dataset since even after 20 hours we didn't get results; this approach is very computationally intensive and as the results shows, there is no benefit in using this method; as an example, as can be seen in Table V, we could get the same degree of 84.46% accuracy from simple kNN in only 40 minutes. Consequently, we believe there is no need to use kernel functions to transfer data to another space.

TABLE VII. RESULTS OF KERNELIZED SVM

#	data points	time	kernel	C	best	Result
1	500	65s	['linear','poly','rbf','sigmoid']	[0.00001,0.0001,0.001,0.01,0.1,1]	kernel='poly', degree=3	60.2
2	5000	62m	['linear','poly', 'rbf','sigmoid']	[0.00001,0.0001,0.001,0.01,0.1,1]	kernel='poly', degree=3	0.7

I. Convolutional Neural Networks (CNN)

Our first model had very poor performance, as it was trained on raw data and had a very simple architecture. Performance of the models generally increased with the number of convolution layers, until it started plateauing at 8 layers for a little above 90% accuracy. From then on, we adjusted batch size and learning rates to achieve better accuracy.

Our most promising model, model 9, has achieved 96.113% accuracy on the final test set on Kaggle. We can observe that the model has comparable performance on the training and validation datasets (see Fig. 2). The accuracy on the validation set follows closely the accuracy on the training set. This is symptomatic of minimal overfitting, possibly underfitting. This is the result we had expected by adding dropout and regularization to the model. However, we can also see that the accuracy was still slightly increasing at the end of the training. Therefore we could have increased the number of epochs to maximize accuracy.

Looking at the overall shape of the model loss in Figure 3, we can see that the drop in model loss is very steep. Also, the validation loss tracks very closely the training loss over the entire training. This could be an indication of underfitting. This may be due to using too much

regularization, which we can fix by changing the batch regularization and dropout in our model. The possible underfitting could also be due to our model being too simple.

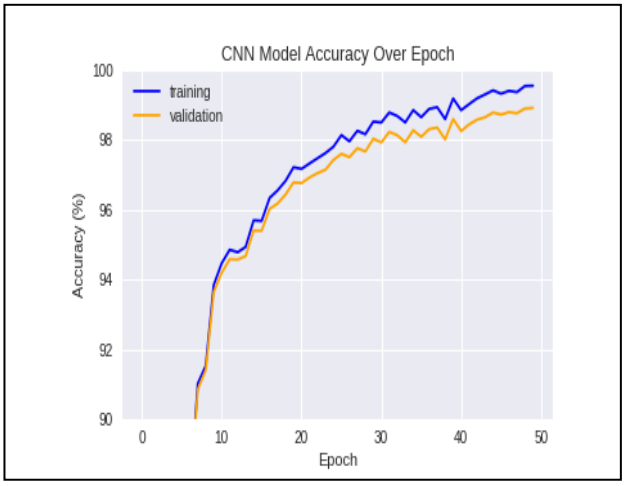


Fig. 3. CNN Model Accuracy Over Epochs

TABLE VIII. CNN HYPERPARAMETER FINE-TUNING

#	Pre-processing	Num. Convolutions	Batch Size	Learning Rate	Acccuracy %
1	No	2	50	0.001	46
2	yes	2	50	0.001	54
3	yes	3	50	0.001	69
4	yes	4	50	0.001	76
5	yes	5	50	0.001	86
6	yes	6	50	0.001	90
7	yes	8	50	0.001	94
8	yes	8	30	0.001	96
9	yes	8	25	0.0001	96
10	yes	9	50	0.001	93

VI. DISCUSSION

As evidenced by the results presented in Section V above, image preprocessing is key to obtaining decent accuracy rates. Parameter initialization may also play a key role as could be seen with FNN. As can be seen in Table IX below, CNN performed best, followed by bagging, random forest and k-NN. In Table IX however, note that only rows in bold show accuracy results on Kaggle dataset. The other accuracy rates are from cross-validation, so may be a lower on test dataset.

TABLE IX. LEARNER ACCURACY

Learner	Accuracy %
CNN	**96.1**
Bagging	90.0
Random forest	87.5
k-NN	82.3
Decision tree	74.5
Linear SVM	73.2
Kernelized SVM	70.0
Boosting	61.5
FNN	**42.5**

As part of image preprocessing, we reduced the size of the images from 64-by-64 to 32-by-32. This accelerated the learning process as already stated, and allowed us to cross-validate and test many more hyperparameter combinations than possible otherwise. However, some relevant features may have been lost in the process. It would be interesting to fine-tune hyperparameter combinations on 64-by-64 images.

More architectures, using more layers could be explored, for all learners we tried. For FNN, alternative activation functions, such as ReLU and tanh, could be tried. Also, improvements could be added to the optimizer, using momentum or adaptive learning rate. Given that cross-validation accuracy is higher than test set accuracy rate for the various learners we tried (more so for some of our learners as FFN), this could indicate overfitting and it could be explored how to reduce overfitting with regularization. Data augmentation could also be a technique worth exploring to help combat overfitting.

In the case of CNN, there are symptoms of underfitting in our model. We could remediate to this issue by further adjusting the learning rate, as it seemed to high. Reducing the intensity of regularization would also help combat underfitting. Another avenue would be to create a more complex architecture by adding more layers, as already stated, or by increasing the number of filters.

VII. STATEMENT OF CONTRIBUTIONS

All team members discussed, took part in defining the problem, and contributed to general sections in report. Then each team member focused on a specific area: image preprocessing (Mehrzad), SVM (Mehrzad), FNN (Yann), Decision Tree, Random Forest, KNN, Bagging and Boosting (Mehrzad), CNN (Philippe for the code that got our best score on Kaggle, combined with Mehrzad's preprocessing; Yann). For each of these areas, each team member did everything from developing the methodology and fine-tuning hyperparameters, to coding the solution, analyzing results, and drafting the relevant part of the report. We hereby state that all the work presented in this report is that of the authors.