Machine Learning Default Project Report

Jeffrey Wen
Hung-Hsiu Yen
Arti Vedula

ECE 5300

1 Problem

This project focuses on applying various machine learning techniques to classify an unknown dataset. The provided dataset contains 100,000 samples each with 20 features. All of the samples are labeled with 1 of 20 class labels ranging from integer values of 0 to 19. The dataset is well-balanced with each class having 5,000 samples. As the given data comes from unknown origins, it is difficult to say what machine learning method should perform the best. Spatially-related data may be best tackled by a convolutional neural network while a random forest might be better suited for clustered data. Thus, without the background knowledge of the data, our task is to apply multiple different machine learning methods to see which algorithm is able to provide the highest classification accuracy during testing.

2 Methods

For the given dataset, we apply several common machine learning techniques to evaluate which method should be adopted for application on unknown testing data. With each method, there is a multitude of design choices that can potentially improve or hinder the performance of the algorithm. While unable to test every possible hyperparameter, we do attempt to find hyperparameters that perform well. To accomplish this, we use several values for each hyperparameter and evaluate the performance of the algorithm with cross-validation. For methods that are quick to train such as logistic regression, we are able to perform k-fold cross-validation. For methods where training can be very time-exhaustive, we perform a cross-validation by splitting the given data into a training set and testing set with an 80/20 split respectively. The features of the data are first standardized to prevent the scale of the features from affecting the outcome of the algorithms. The details of each technique are described in the following section. We specify the individual contributions of each member of our team in the section header.

2.1 Logistic Regression (Hung-Hsiu Yen)

Compare with linear regression, logistic regression solve the problem of imbalanced data by applying likelihood function (logistic function) to the score. In the classifier setting, we could set `multi_class` as `"multinomial"` for jointly designing parameters which is comparing the class one by one and `multi_class` as `"ovr"` for one versus rest design. Because the former is comparing one by one, it needs more computational power than the later.

In the training, we first use the jointly training with default parameters and compare the results with one versus all training. Since the L2 regularization is the build in default in logistic regression in sklearn package, we use the L1 regularization in one versus rest classification to tune and check whether there's an improvement in classification. During data processing, we found that the computing time increased when we standardized the data, thus I switch the K-folds validation to 80/20 split to increase the traning efficiency.

2.2 Support Vector Classifier (Hung-Hsiu Yen)

In the logistic regression, we consider all the data during the classification, but in some situation we can consider using part of the data to do the classification. The support vector classifier (SVC) focuses only on the samples (support vectors) that determines the hyperplane that classifiers the dataset. Similar to logistic regression, SVC has jointly design and one versus all design.

First, we apply the jointly training to dataset and compare the results with one versus all training with default parameters. Since the L2 regularization is also the build in default, we use the L1 regularization in one versus rest classification to tune and check whether there's an improvement in classification. In the training , we also use 80/20 split for the traning efficiency.

2.3 Support Vector Machine (Hung-Hsiu Yen)

Some datasets are not lineary separable, the SVC approach would go wrong. No matter how you tune the parameters, there are some data that violates the constraints. Thus we could use some transformation functions to transform the data to a new features to get linearly separable. But the transformation functions usually have infinite dimensions, we use kernel tricks to make them become finite dimensions which is easier to compute. There are some popular kernels could be used such as radial basis function(RBF), polynomial, sigmodal...etc

In the training, we apply the jointly training to dataset and compare the results with one versus all training with default parameters. To optimize the C and gamma parameters, we observe the default parameters and guess 9 combinations to find optimal value for jointly training.

2.4 Two-Layer Neural Network (Jeffrey Wen)

While other methods like SVM perform a specified feature transformation, we would like to see if we can learn a better feature transformation with neural networks. We first start with a shallow, two-layer neural network. The neural network consists of an input layer, hidden layer with an activation function, and an output layer. As we are performing 20-way classification, we utilize the cross-entropy loss with an Adam [3] optimizer. Since the PyTorch implementation of the cross-entropy loss exhibits the logits input, we do not add an activation function to the output of the network.

With just this foundation, there are many different design choices to consider. Since there are so many options to try and neural networks can take quite some time to train, we only use a cross-validation with a random 80/20 split between training and testing data. For each hyperparameter we tested, we took the value which provided the largest testing accuracy. We attempt multiple values for the number of hidden nodes and learning rate. We also test three different activation functions: sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Batch normalization is a technique which standardizes the outputs of the hidden layer before the activation function to a learned mean and standard deviation. This additional layer has been known to allow for higher learning rates. Dropout is a technique for preventing overfitting by removing a subset of hidden nodes randomly during each minibatch of training. The nodes are removed with a probability of 0.5

in our implementation. To evaluate the performance gains or losses of these additions, we test our network with different combinations of including or excluding batch normalization and dropout. Each variation of our network is trained for 10 epochs.

2.5 Convolutional Neural Network (Jeffrey Wen)

Convolutional Neural Networks (CNNs) generally perform well on data with spatial relations such as images or time-series data. This is because CNNs use individual kernels that are convolved with the input similar to using a Finite Impulse Response (FIR) filter on an electric signal. The kernels allow the CNN to detect local features in the input and then combine those features in later layers to define more complicated features. By passing the kernels over the entire input, the detected features are translation-invariant. This allows CNNs to perform well with tasks such as classifying objects in an image where the object could be anywhere in the image.

We set up our CNN with two main groups of layers. The first group is a sequence of one-dimensional convolutional layers. For simplicity, each convolutional layer has the same number of output channels, but we test multiple values to find the best performing number of channels. Each convolution layer includes an activation function and uses the same-padding mode with a stride of 1. After the convolutional layers, the output is flattened into a vector and passed the the second group of dense layers. The dense layers perform exactly as they do for the 2-layer neural network described above. Again, we use the cross-entropy loss with an Adam optimizer, so the output of the dense layer group does not need an activation function. For each dense layer added, we divide the number of hidden layers in half.

Using cross-validation with the 80/20 split between training and testing data, we evaluate the network to find well-performing choices for the type of activation function, number of convolutional layers, number of dense layers, learning rate, kernel size, number of channels in each convolution layer, and the inclusion or exclusion of batch normalization and dropout in the convolutional layers. For each hyperparameter, we take the configuration that provides the highest testing accuracy. Batch normalization and dropout are only applied to the convolutional layers in this setup.

2.6 Random Forest (Arti Vedula)

Random forest methods[4] are ensemble learning methods that can be used for classification. These methods work by constructing many simple decision trees for different subsets of the data during train time, and averaging several of these weak classifiers to make a stronger prediction during test time. Random forest methods are preferred over computationally heavy methods[5], such as neural networks or deep methods as they provide comparable results however are relatively easy to train. One of the biggest advantages of this method is that it is easy to understand.

We start out by initializing our random forest to contain 100 decision trees, as a default value, to measure the baseline accuracy over standardized training features. We then fit the data to our model using k-fold cross validation. We tune the parameters of the random forest in order to determine experimentally the optimum values for our given dataset. We evaluate the model at different combinations of estimators ranging from 100 to 1000, depths ranging between 10 to 100 and bootstrap values of True, and False to determine whether or not to replace a value back into our training pool.

2.7 Gradient Boosting(Arti Vedula)

Gradient Boosting[2] is a re-interpretation of AdaBoost, a popularly used boosting method. It is a sequential training method. These methods are preferred over computationally heavy methods, such as neural networks or deep methods as they provide comparable results however are relatively easy to train. These methods use a second order Taylor series expansion, and also use parallelization and hardware acceleration tricks.

We use this method to simply serve as a baseline accuracy of about 90% for our XGBoost methods and thus initialize our classifier to contain the default values of 100 decision trees with depth 3 over standardized training features.

2.8 Extreme Gradient Boosting (Arti Vedula)

Extreme Gradient Boosting [1] is an ensemble learning method that can be used for classification. It is an evolution of the Gradient Boosting method as defined above. These methods are preferred over computationally heavy methods, such as neural networks or deep methods as they provide comparable results however are relatively easy to train. These methods use a second order Taylor series expansion, and also use parallelization and hardware acceleration tricks.

We start out by initializing our classifier to contain the default values, to measure the baseline accuracy over standardized training features. We then fit the data to our model using k-fold cross validation. We tune the parameters of the XGBoostClassifier in order to determine experimentally the optimum values for our given dataset. We evaluate the model at different combinations of estimators ranging from 100 to 1000, depths ranging between 1 to 10 and learning rate values ranging from 0.01 to 0.1.

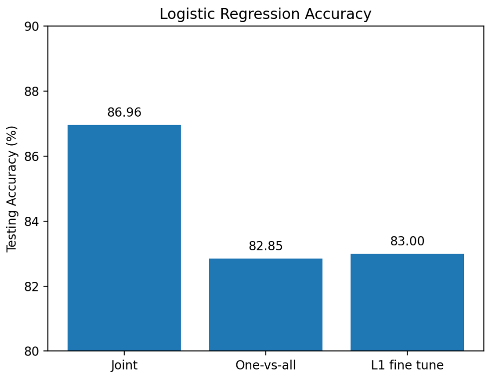
3 Results

3.1 Logistic Regression

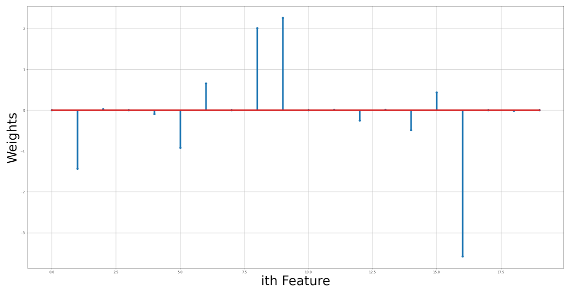
The result of different methods is shown in Fig. 1. In Fig. 1a, the joint classification shows testing accuracy of 86.96% which is better than one versus all at the same parameter setting. This is because joint design compares data with each class one by one instead of comparing one class versus rest of the class. In L1 regularization, the testing accuracy of one versus all is improved by adjusting penalty parameter c. From part of weights shown in Fig. 1b, lots of weights are compressed to zero, meaning that some features in the data set have minor contribution for class estimation. The confusion of best testing accuracy (Joint) is shown in Appendix Fig. 12. Class 8, 10, 13, 14 show relative lower matching correctness. 17% of Class 10 are misclassified with Class 12 and large amount of Class 13, Class 14 are misclassified with Class 8 by a percentage of 8.2% and 17%.

3.2 Support Vector Classifier

The result of SVC is shown in Fig. 2. From Fig. 2a, similar to the logistic regression, a better testing accuracy of 87.09% is obtained from joint design, owning to the way it classifies data. Little improvement of L1 regularization could be observed. This may due to the reason that SVC only choose the part of the data that has contribution in hyperplane determination, thus less improvement could be seen for silencing the features of these support vectors.From confusion matrix in Appendix Fig. 13, similar to logistic, Class 8, 10, 13, 14 show relative lower matching correctness. Class 10 are misclassified with Class 12 and large amount of Class 13, Class 14 are misclassified with Class 8.



(a) Testing Accuracy of Different Setting



(b) Parts of Weights from L1 Regularization

Figure 1: Testing accuracy of logistic regression with different methods

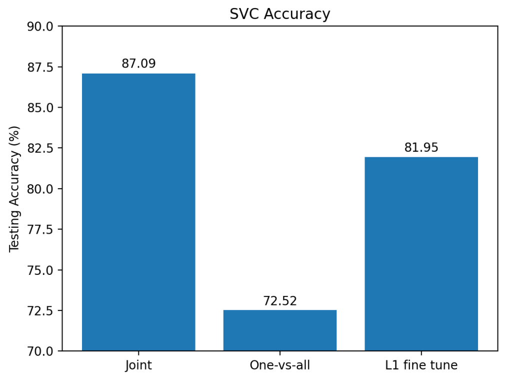
3.3 Support Vector Machine

In the result of SVM Fig. 2a, compare with SVC, a slightly improve can be observed. This may due to the reason that the data is transformed into another formation that is more linear separable than SVC. Because the default value of gamma is inverse value of multiplication between numbers of features and data variance, optimal parameters are tuned based on the value and a accuracy of 90.1% is achieved.From confusion matrix in Appendix Fig. 14,Class 8, 10, 13, 14 show relative lower matching correctness.

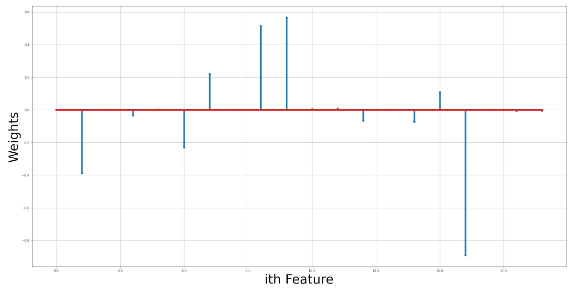
3.4 Two-Layer Neural Network

The results of all the cross-validation tests with different hyperparameters can be seen in the plots of Fig. 4. As shown in Fig. 4a, 1000 nodes in the hidden layer provided the best testing accuracy of 88.19%; however, the performance gains over 50, 100, and 500 hidden node models was marginal. The best performing activation function was the ReLU function at 90.55%. Using neither dropout or batch normalization gave the largest testing accuracy at 90.81%. The inclusion of a dropout or batch normalization layer only seemed to hurt the performance slightly while the inclusion of both gave a dip of more than 2 percentage points. Fig. 4(d) shows the testing performance during each training epoch. A learning rate of 0.001 provides the highest accuracy while giving a smooth plot.

With all of the top performing hyperparameters from above, the network achieves a testing accuracy of 90.74% and training accuracy of 91.70%. The confusion matrix for the testing set is shown in Fig. 15 in the Appendix A. Classes 10 and 12 seem to get confused with one another the most with 17% of class 10 samples being classified as class 12 and 14% the other way. The network

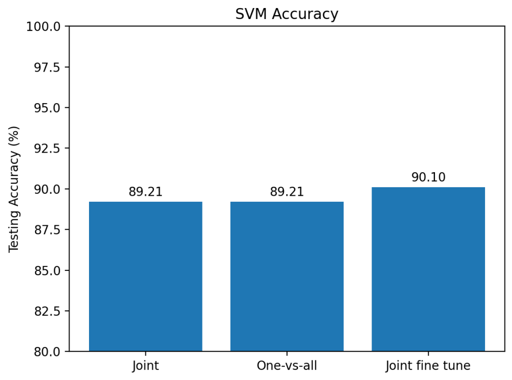


(a) Testing Accuracy of Different Setting



(b) Parts of Weights from L1 Regularization

Figure 2: Testing accuracy of SVC with different methods



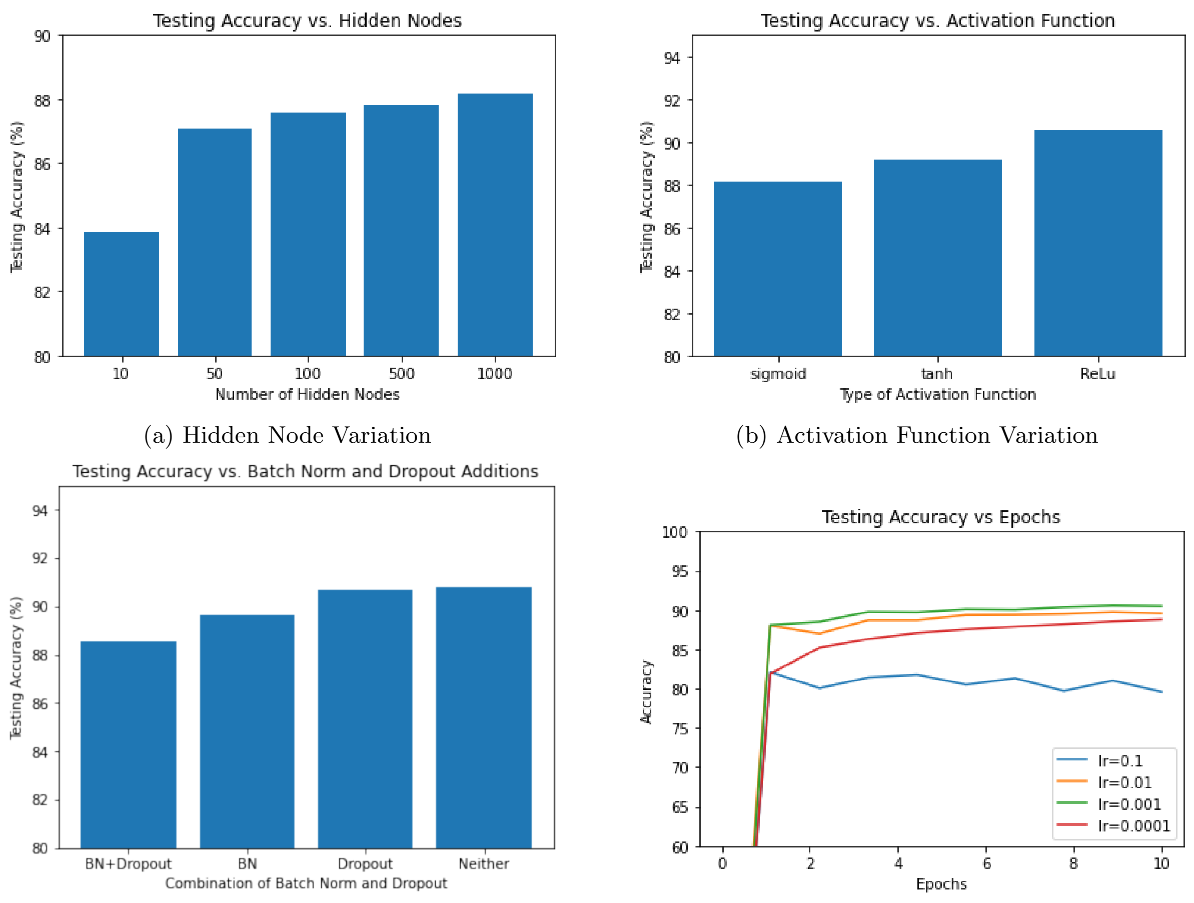
(a) Testing Accuracy of Different Setting

Figure 3: Testing accuracy of SVM with different methods

also struggles with classes 8 and 14, but it more consistently labels class 14 samples as class 8 rather than the other way around. When trained with all the given data, the network provides a 91.72% training accuracy.

3.5 Convolutional Neural Network

The results of varying different hyperparameters are show in Fig. 5. Many of the values for some of the hyperparameters seemed to make very little difference in the testing accuracy. For the activation



(a) Hidden Node Variation

(b) Activation Function Variation

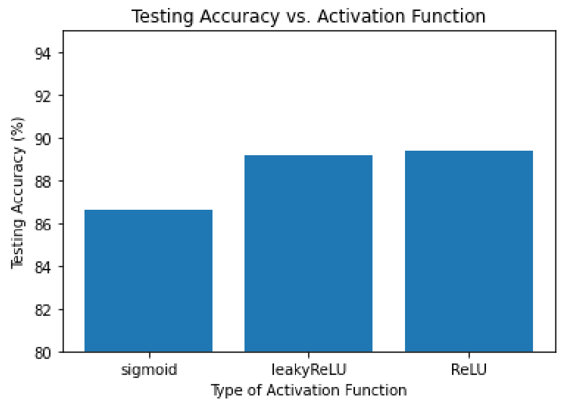
(c) Batch Norm and Dropout

(d) Learning Rate Variation

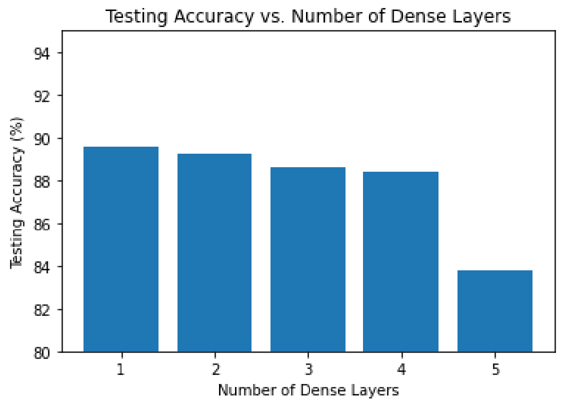
Figure 4: Testing accuracy of 2-layer neural network with different hyperparameters

function, either leakyReLU or ReLU would give comparable performance to one another with ReLU providing a slightly higher test accuracy at 89.40% in our test. The difference between 1, 5, and 10 convolution layers was also very small with the slight advantage given to 1 convolution layer. The performance was best with 1 dense layer as well. Fig 5c shows that a learning rate of 0.001 provides the highest accuracy and gives a very smooth plot. Kernel size makes very little difference in the performance with the 3x3 kernel giving the top score of 89.75%. Using 7 channels between the convolutional layers provides the best performance at 89.97%. The network does not seem to benefit from the inclusion of batch normalization or dropout although batch normalization does not seem to hurt the performance much. For configurations where the change in hyperparameter yielded very little difference, the plots were not included.

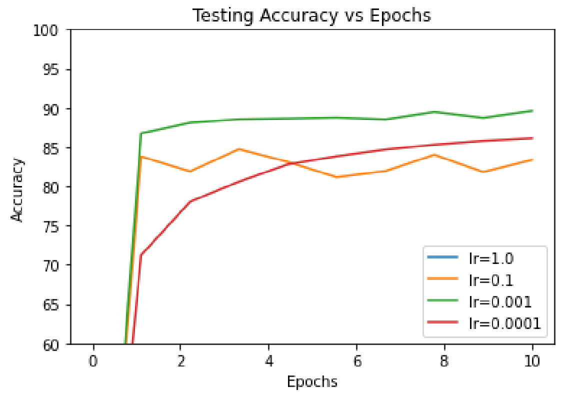
Looking at the confusion matrix on the testing set for the final CNN with the top performing hyperparameters, the model seems to struggle with the same classes as the 2-layer neural network, namely classes 10 and 12 and classes 8 and 14. The confusion matrix can be seen in Fig. 16 of Appendix A. The overall testing accuracy of this configuration is 90.08%. Training this configuration on the entire dataset yields a training accuracy of 90.52%.



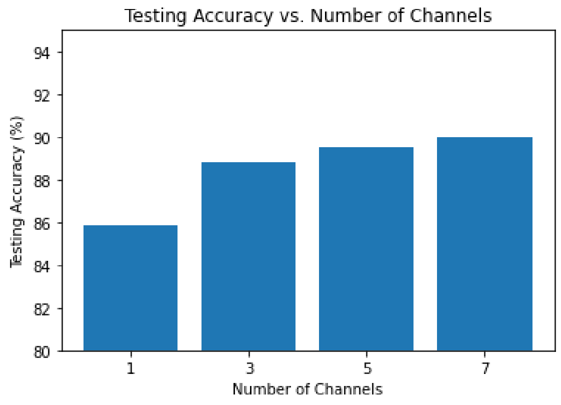
(a) Activation Function



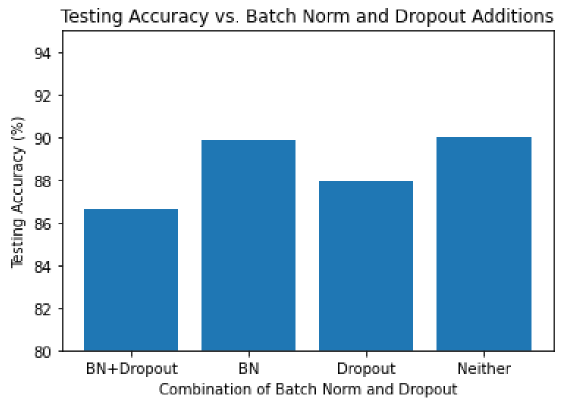
(b) Number of Dense Layers



(c) Learning Rate



(d) Number of Channels



(e) Batch Norm and Dropout

Figure 5: Testing accuracy of convolutional neural network with different hyperparameters. Note: for (c), there are configurations with accuracy too low to be visible in the plots.

3.6 Random Forest

The results of the cross validation tests are shown in the figures given below Fig. 6. We see that 500 estimators provided the best testing accuracy of 90.13%, however over all the estimators, there isn't a huge increase in performance as the number of estimators increase. As our network begins to balloon with increasing depth, there is also the danger of overfitting, as a result, we decide to go with a more modest depth of 10, in the hopes that our trees can generalize well to unseen data. We then get a final accuracy of 87.2%. The variation of accuracy across various hyperparameters is shown below. We search for the optimal solution using RandomizedSearchCV for a combination of estimator values, depth values and bootstrap values, using 10-fold cross validation.

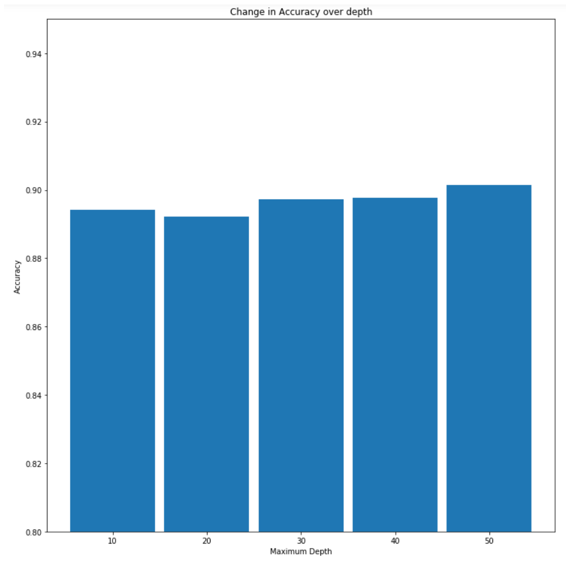


Figure 6: Testing accuracy of Random Forests as a function of the depth

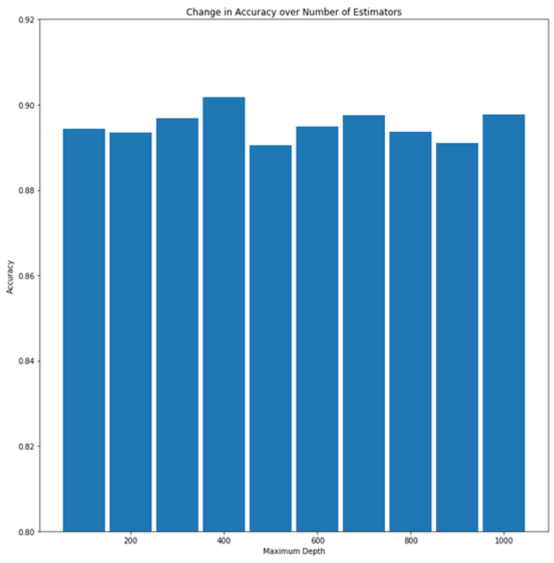


Figure 7: Testing accuracy of Random Forests as a function of the number of estimators

3.7 Extreme Gradient Boosting

The results of the 10-fold cross validation tests are shown in the figures given below Fig. 9. We see that 100 estimators provided the best testing accuracy of 98%, over all combinations. We control the depth strictly here, and get a maximum depth of 6, and a learning rate of 0.3. We then get a final accuracy of 98.7%. The variation of accuracy across various hyperparameters is shown below. We search for the optimal solution using RandomizedSearchCV for a combination of number of estimators, depth values and learning rates, using 10-fold cross validation.

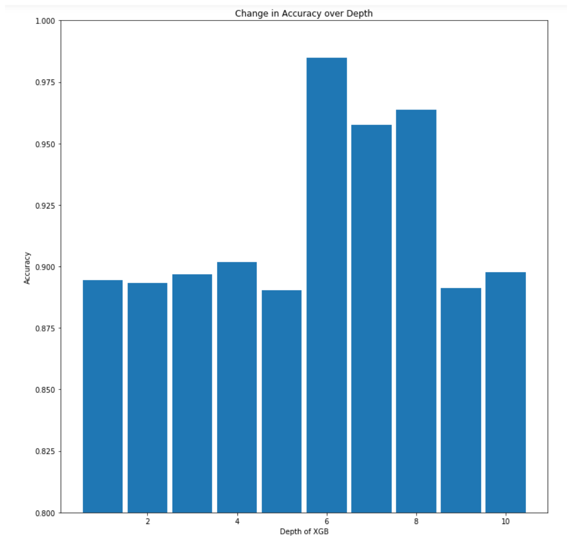


Figure 8: Testing accuracy of XGBClassifier as a function of the depth of network

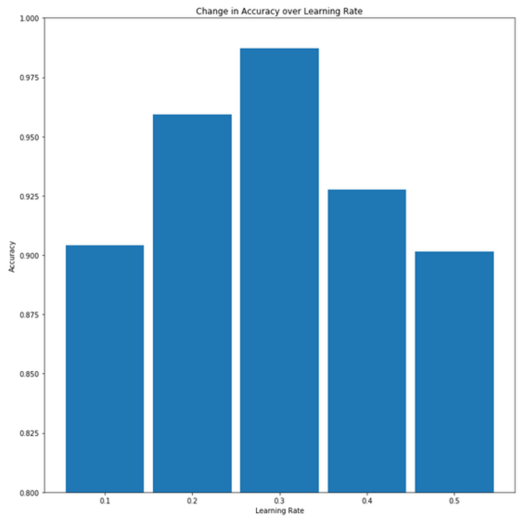


Figure 9: Testing accuracy of XGBClassifier as a function of the learning rate

4 Analysis

In comparison between logistic regression and SVC/SVM, the logistic regression is defining a probabilistic model based on all the dataset that maximizes the likelihood. Therefore, when the number of training sample becomes large, it will have trouble in matching all the training samples from each class just like the model is overfitting. On the other hand, the SVC/SVM only use some important data as support vectors to find a maximum boundary across each class. Thus, compare with the logistic regression, the SVC/SVM could have less overfitting effect that could have better prediction. In SVM, the data set could be transformed to another dimension to make it become more linearly separable which improves the accuracy rate.

Analyzing the two neural network approaches, we can see that the 2-layer neural network provides a higher testing accuracy and training accuracy on the final configurations compared to the CNN. This may be because the data is not spatially related, and thus, there is very little benefit to performing convolutions. We also observed that the CNNs of larger depth both with the convolutional layers and dense layers actually performed worse than having a single convolutional layer and single dense layer. The data may simply not require a network of such large capacity. As discussed in class, layers can struggle to act like an identity, so if the network is deeper than required, it can perform worse than a shallower network.

When we use random forests, one of the major disadvantages of the method is that it is highly prone to overfitting, and if we are not careful, it can lead to an extremely deep ensemble of trees, but the generalization capabilities of the network will be quite poor. When we aren't aware of our data distribution, it is more important that we don't carelessly overfit to what we have without appropriate domain knowledge. XGBoost uses the regularization parameter to safeguard us against this overfitting, and as a result, we see that the XGBoost gives better results on the training data than Random Forests, and it is faster to train, and can be parallelized on a GPU.

We have put the confusion matrices for each method in Appendix A. The confusion matrices demonstrate that all of the methods seem to struggle with the same set of classes. Class 10 and 12 are commonly misclassified as the other class, and class 8 and 14 have a similar problem. This indicates that these sets of classes may share a common set of features that makes them hard to distinguish. While its difficult to say what the commonality of these classes may be, the consistent performance gap throughout all of the machine learning methods does allow us see connections between classes of data that we know nothing about.

We visualize the distribution in the feature space as shown in fig 10, using the t-SNE distribution, and observe, that the machine learning methods we have used also form similar conclusions to our data visualization. The visualization shows a significant overlap in categories 8,14 and 10,12, which is understandably where our algorithm trips up. It seems there is a strong correlation between these two features. We also notice that feature 0 is more spread out, which again is a conclusion that we can draw from our confusion matrix. Thus, we see that our machine learning methods perform decently well in approximating the distribution of the data that has been provided to us. We see, that among all methods, XGBoost performs the best, whereas Logistic Regression seems to perform poorly, as can be seen in Fig. 11.

References

[1] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4, 2015.

[2] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[4] Mahesh Pal. Random forest classifier for remote sensing classification. *International journal of remote sensing*, 26(1):217–222, 2005.

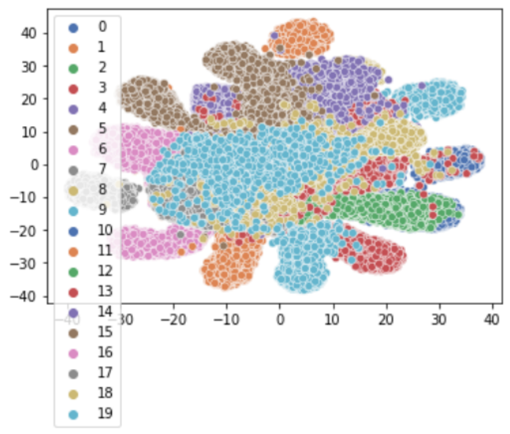


Figure 10: t-SNE Data Visualization of 20 categories in the feature space

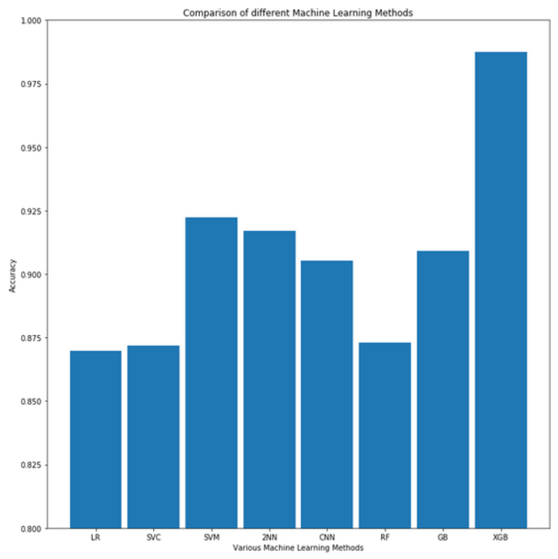


Figure 11: Comparison of accuracies of all methods on the test set

[5] Victor Francisco Rodriguez-Galiano, Bardan Ghimire, John Rogan, Mario Chica-Olmo, and Juan Pedro Rigol-Sanchez. An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 67:93–104, 2012.

5 Appendix A: Confusion matrices

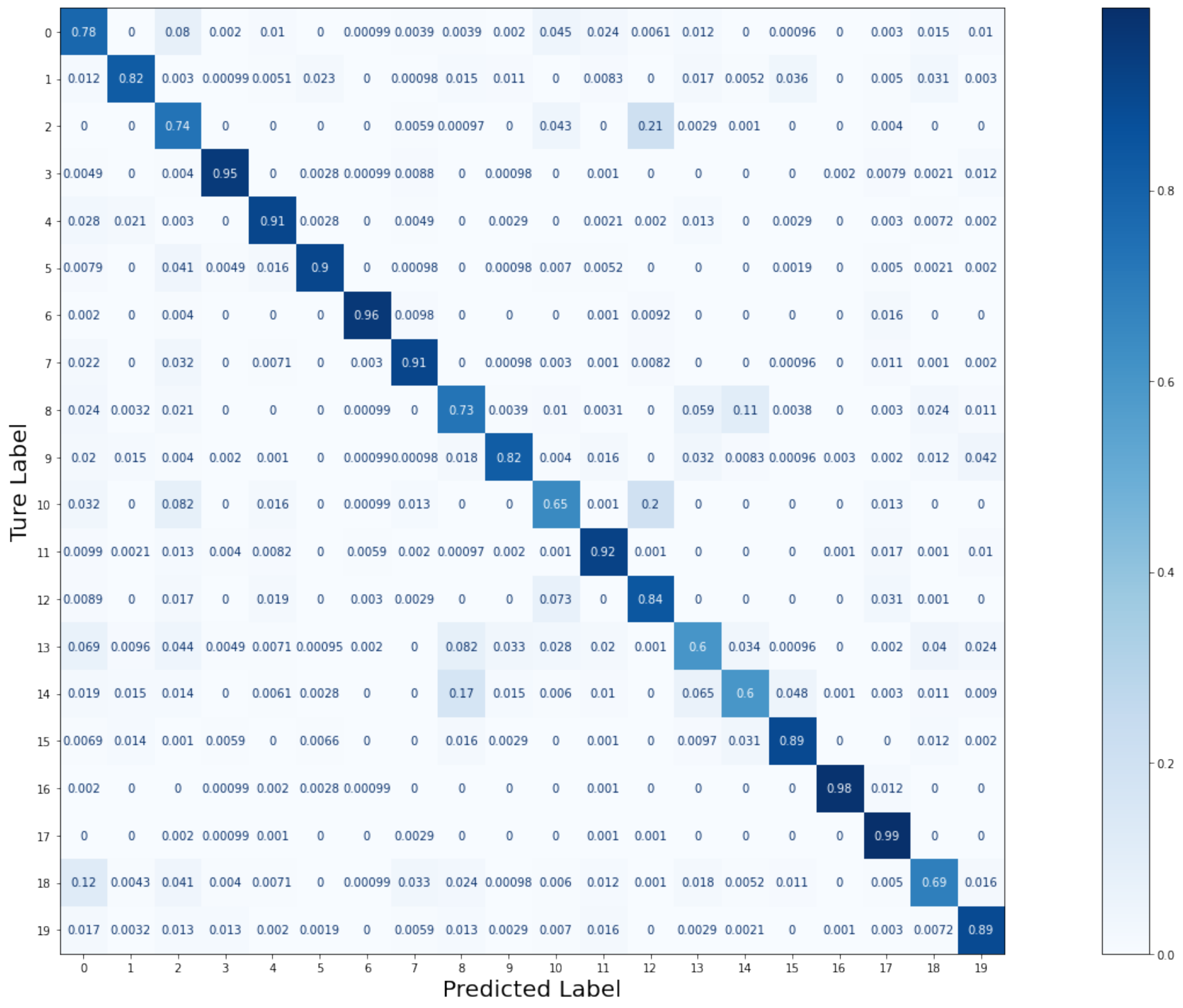


Figure 12: Confusion matrix for the Logistic Regression on test set

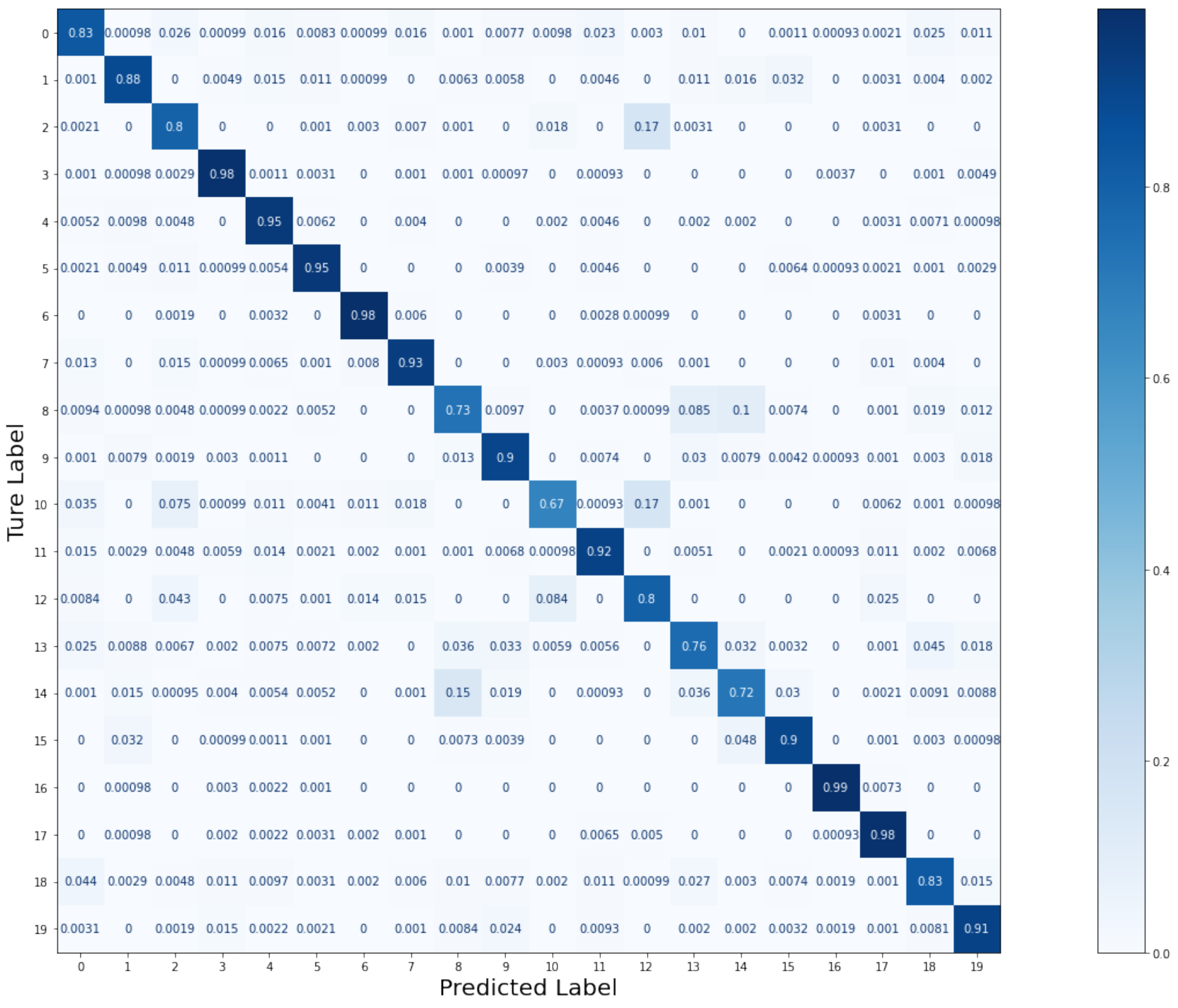
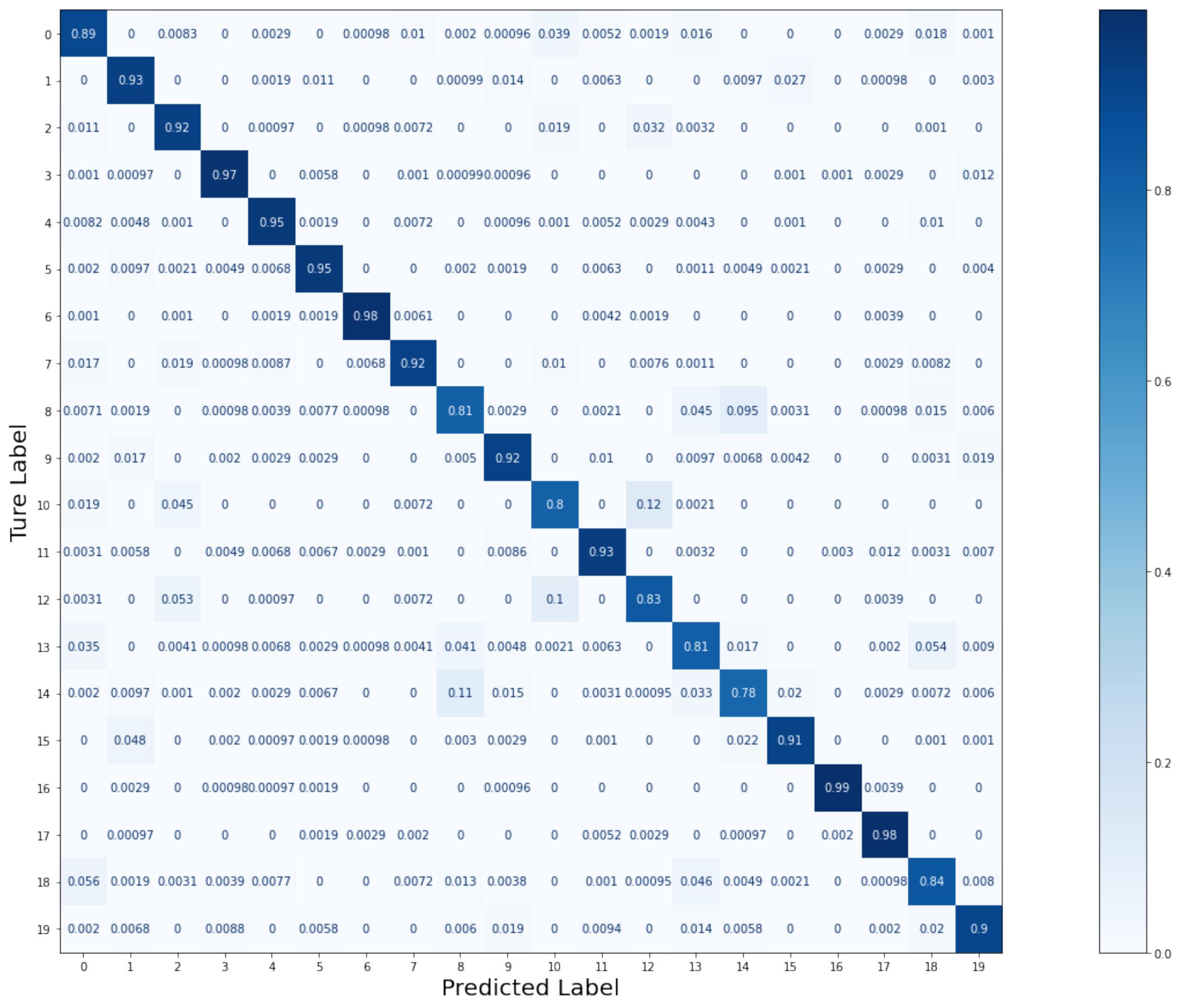


Figure 13: Confusion matrix for the SVC on test set



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0.89	0	0.0083	0	0.0029	0	0.00098	0.01	0.002	0.00096	0.039	0.0052	0.0019	0.016	0	0	0	0.0029	0.018	0.001
1	0	0.93	0	0	0.0019	0.011	0	0	0.00099	0.014	0	0.0063	0	0	0.0097	0.027	0	0.00098	0	0.003
2	0.011	0	0.92	0	0.00097	0	0.00098	0.0072	0	0	0.019	0	0.032	0.0032	0	0	0	0	0.001	0
3	0.001	0.00097	0	0.97	0	0.0058	0	0.001	0.00099	0.00096	0	0	0	0	0	0.001	0.001	0.0029	0	0.012
4	0.0082	0.0048	0.001	0	0.95	0.0019	0	0.0072	0	0.00096	0.001	0.0052	0.0029	0.0043	0	0.001	0	0	0.01	0
5	0.002	0.0097	0.0021	0.0049	0.0068	0.95	0	0	0.002	0.0019	0	0.0063	0	0.0011	0.0049	0.0021	0	0.0029	0	0.004
6	0.001	0	0.001	0	0.0019	0.0019	0.98	0.0061	0	0	0	0.0042	0.0019	0	0	0	0	0.0039	0	0
7	0.017	0	0.019	0.00098	0.0087	0	0.0068	0.92	0	0	0.01	0	0.0076	0.0011	0	0	0	0.0029	0.0082	0
8	0.0071	0.0019	0	0.00098	0.0039	0.0077	0.00098	0	0.81	0.0029	0	0.0021	0	0.045	0.095	0.0031	0	0.00098	0.015	0.006
9	0.002	0.017	0	0.002	0.0029	0.0029	0	0	0.005	0.92	0	0.01	0	0.0097	0.0068	0.0042	0	0	0.0031	0.019
10	0.019	0	0.045	0	0	0	0	0.0072	0	0	0.8	0	0.12	0.0021	0	0	0	0	0	0
11	0.0031	0.0058	0	0.0049	0.0068	0.0067	0.0029	0.001	0	0.0086	0	0.93	0	0.0032	0	0	0.003	0.012	0.0031	0.007
12	0.0031	0	0.053	0	0.00097	0	0	0.0072	0	0	0.1	0	0.83	0	0	0	0	0.0039	0	0
13	0.035	0	0.0041	0.00098	0.0068	0.0029	0.00098	0.0041	0.041	0.0048	0.0021	0.0063	0	0.81	0.017	0	0	0.002	0.054	0.009
14	0.002	0.0097	0.001	0.002	0.0029	0.0067	0	0	0.11	0.015	0	0.0031	0.00095	0.033	0.78	0.02	0	0.0029	0.0072	0.006
15	0	0.048	0	0.002	0.00097	0.0019	0.00098	0	0.003	0.0029	0	0.001	0	0	0.022	0.91	0	0	0.001	0.001
16	0	0.0029	0	0.00098	0.00097	0.0019	0	0	0	0.00096	0	0	0	0	0	0	0.99	0.0039	0	0
17	0	0.00097	0	0	0	0.0019	0.0029	0.002	0	0	0	0.0052	0.0029	0	0.00097	0	0.002	0.98	0	0
18	0.056	0.0019	0.0031	0.0039	0.0077	0	0	0.0072	0.013	0.0038	0	0.001	0.00095	0.046	0.0049	0.0021	0	0.00098	0.84	0.008
19	0.002	0.0068	0	0.0088	0	0.0058	0	0	0.006	0.019	0	0.0094	0	0.014	0.0058	0	0	0.002	0.02	0.9

Figure 14: Confusion matrix for the SVM on test set

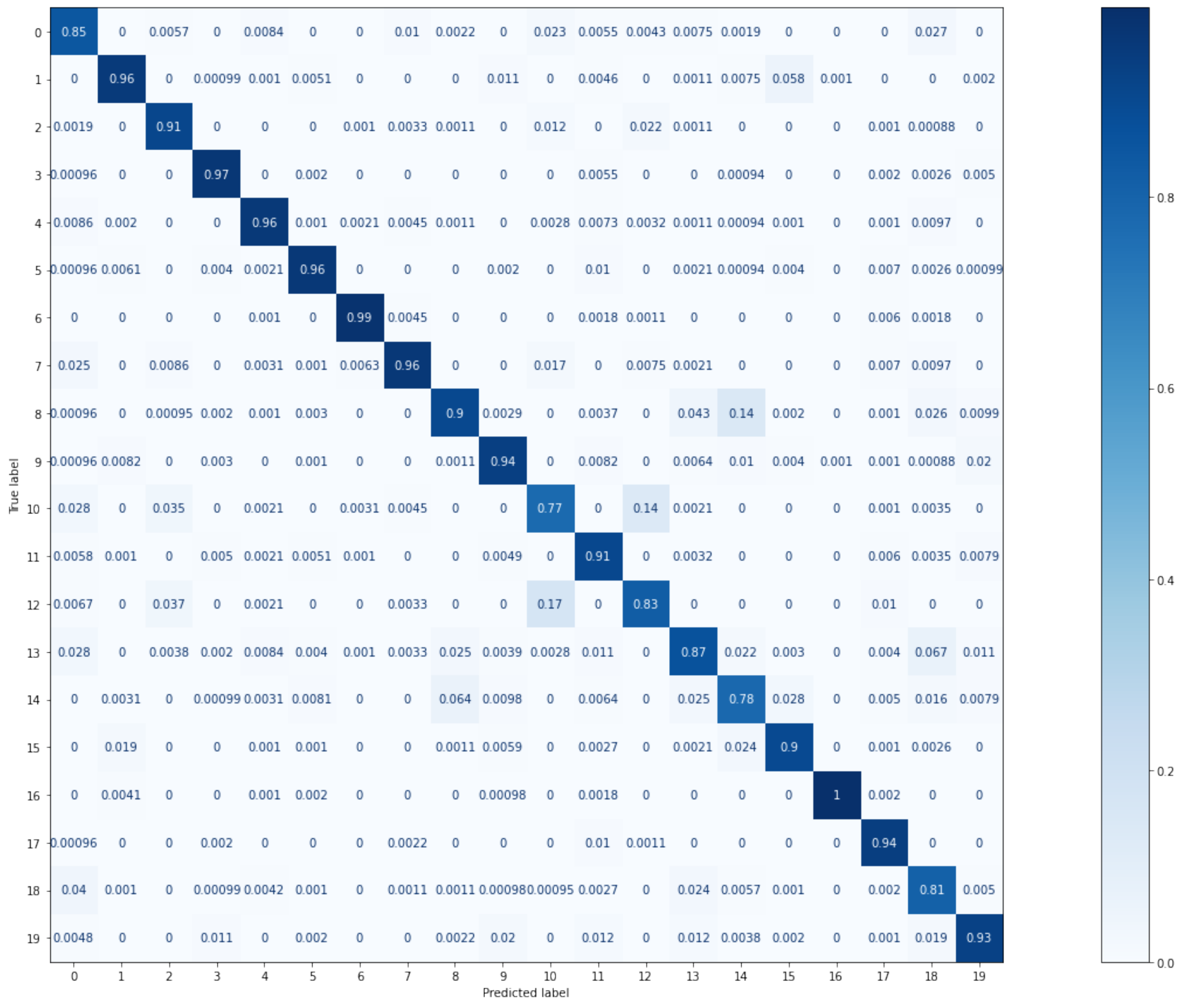
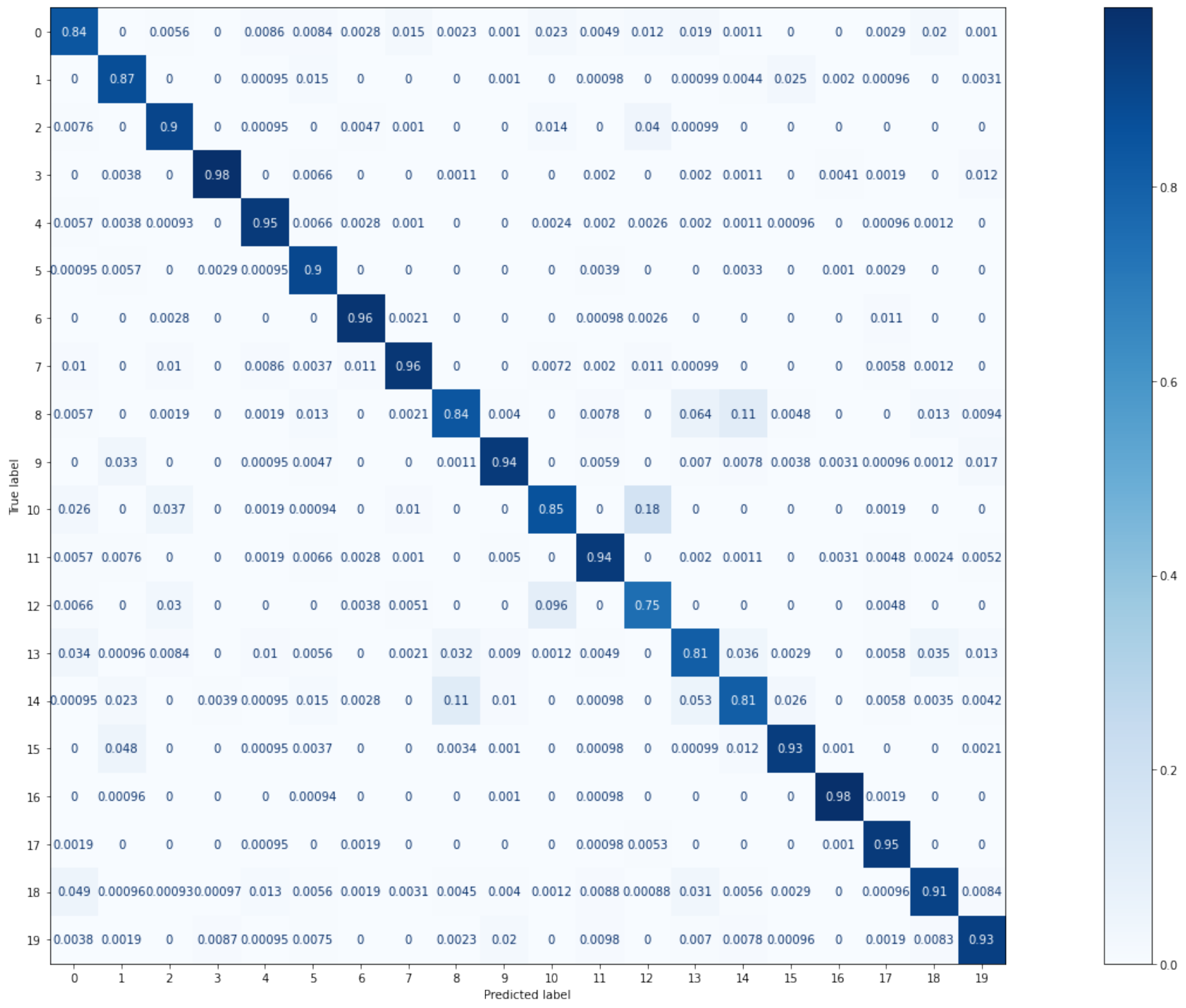


Figure 15: Confusion matrix for the final 2-layer neural network on test set



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0.84	0	0.0056	0	0.0086	0.0084	0.0028	0.015	0.0023	0.001	0.023	0.0049	0.012	0.019	0.0011	0	0	0.0029	0.02	0.001
1	0	0.87	0	0	0.00095	0.015	0	0	0	0.001	0	0.00098	0	0.00099	0.0044	0.025	0.002	0.00096	0	0.0031
2	0.0076	0	0.9	0	0.00095	0	0.0047	0.001	0	0	0.014	0	0.04	0.00099	0	0	0	0	0	0
3	0	0.0038	0	0.98	0	0.0066	0	0	0.0011	0	0	0.002	0	0.002	0.0011	0	0.0041	0.0019	0	0.012
4	0.0057	0.0038	0.00093	0	0.95	0.0066	0.0028	0.001	0	0	0.0024	0.002	0.0026	0.002	0.0011	0.00096	0	0.00096	0.0012	0
5	0.00095	0.0057	0	0.0029	0.00095	0.9	0	0	0	0	0	0.0039	0	0	0.0033	0	0.001	0.0029	0	0
6	0	0	0.0028	0	0	0	0.96	0.0021	0	0	0	0.00098	0.0026	0	0	0	0	0.011	0	0
7	0.01	0	0.01	0	0.0086	0.0037	0.011	0.96	0	0	0.0072	0.002	0.011	0.00099	0	0	0	0.0058	0.0012	0
8	0.0057	0	0.0019	0	0.0019	0.013	0	0.0021	0.84	0.004	0	0.0078	0	0.064	0.11	0.0048	0	0	0.013	0.0094
9	0	0.033	0	0	0.00095	0.0047	0	0	0.0011	0.94	0	0.0059	0	0.007	0.0078	0.0038	0.0031	0.00096	0.0012	0.017
10	0.026	0	0.037	0	0.0019	0.00094	0	0.01	0	0	0.85	0	0.18	0	0	0	0	0.0019	0	0
11	0.0057	0.0076	0	0	0.0019	0.0066	0.0028	0.001	0	0.005	0	0.94	0	0.002	0.0011	0	0.0031	0.0048	0.0024	0.0052
12	0.0066	0	0.03	0	0	0	0.0038	0.0051	0	0	0.096	0	0.75	0	0	0	0	0.0048	0	0
13	0.034	0.00096	0.0084	0	0.01	0.0056	0	0.0021	0.032	0.009	0.0012	0.0049	0	0.81	0.036	0.0029	0	0.0058	0.035	0.013
14	0.00095	0.023	0	0.0039	0.00095	0.015	0.0028	0	0.11	0.01	0	0.00098	0	0.053	0.81	0.026	0	0.0058	0.0035	0.0042
15	0	0.048	0	0	0.00095	0.0037	0	0	0.0034	0.001	0	0.00098	0	0.00099	0.012	0.93	0.001	0	0	0.0021
16	0	0.00096	0	0	0	0.00094	0	0	0	0.001	0	0.00098	0	0	0	0	0.98	0.0019	0	0
17	0.0019	0	0	0	0.00095	0	0.0019	0	0	0	0	0.00098	0.0053	0	0	0	0.001	0.95	0	0
18	0.049	0.00096	0.00093	0.00097	0.013	0.0056	0.0019	0.0031	0.0045	0.004	0.0012	0.0088	0.00088	0.031	0.0056	0.0029	0	0.00096	0.91	0.0084
19	0.0038	0.0019	0	0.0087	0.00095	0.0075	0	0	0.0023	0.02	0	0.0098	0	0.007	0.0078	0.00096	0	0.0019	0.0083	0.93

Figure 16: Confusion matrix for the final convolutional neural network on test set

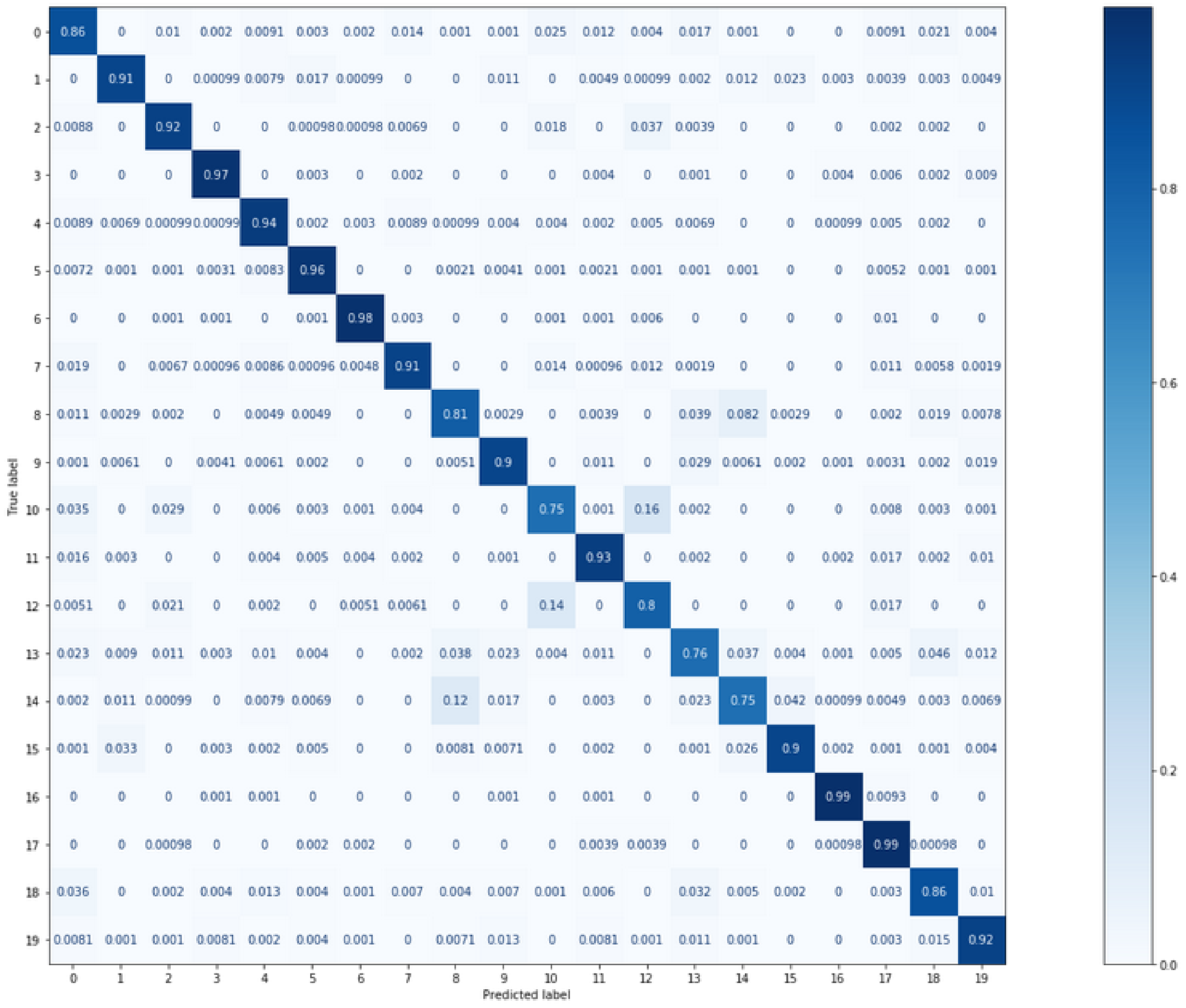


Figure 17: Confusion Matrix for Random Forest on test set

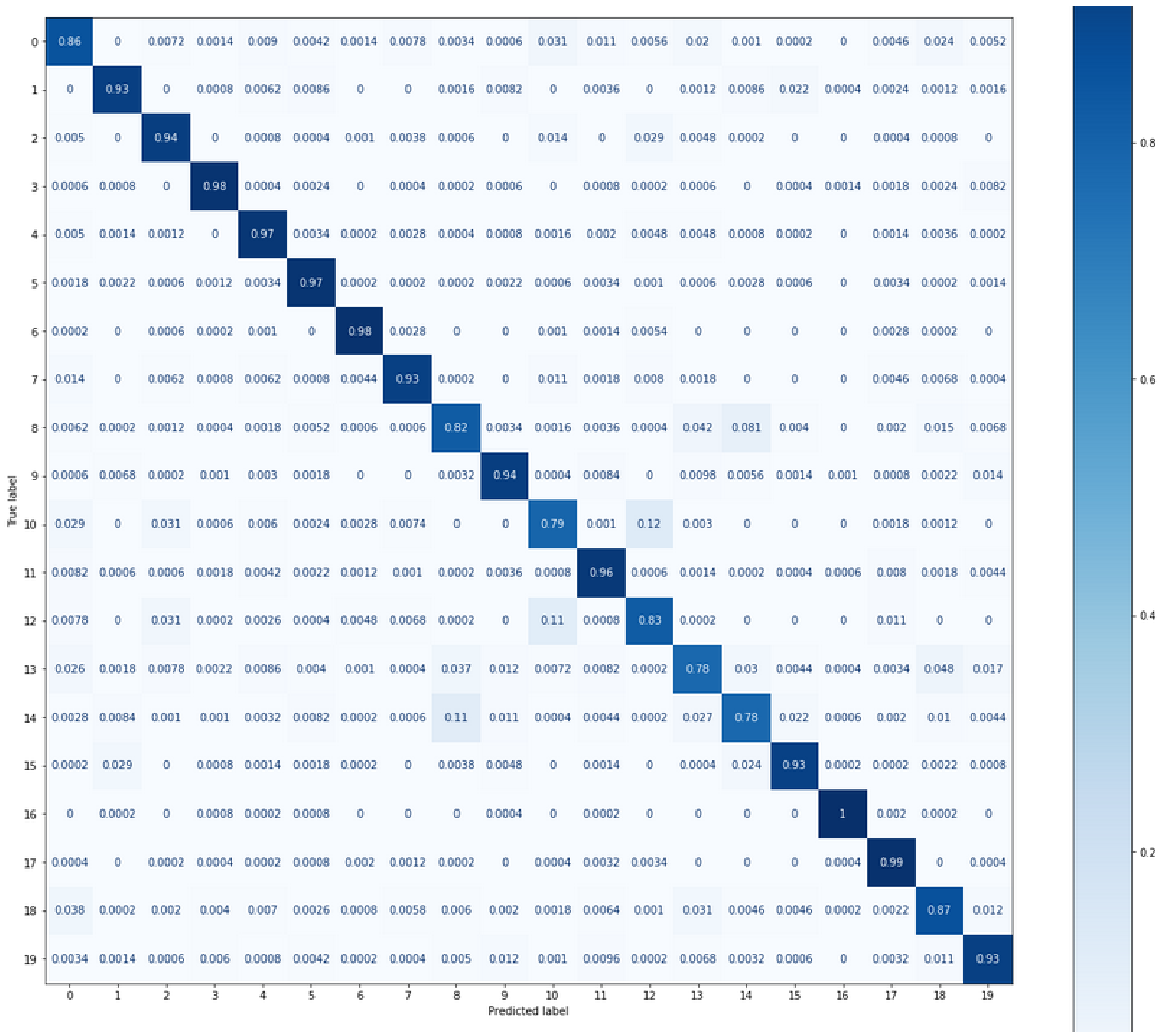
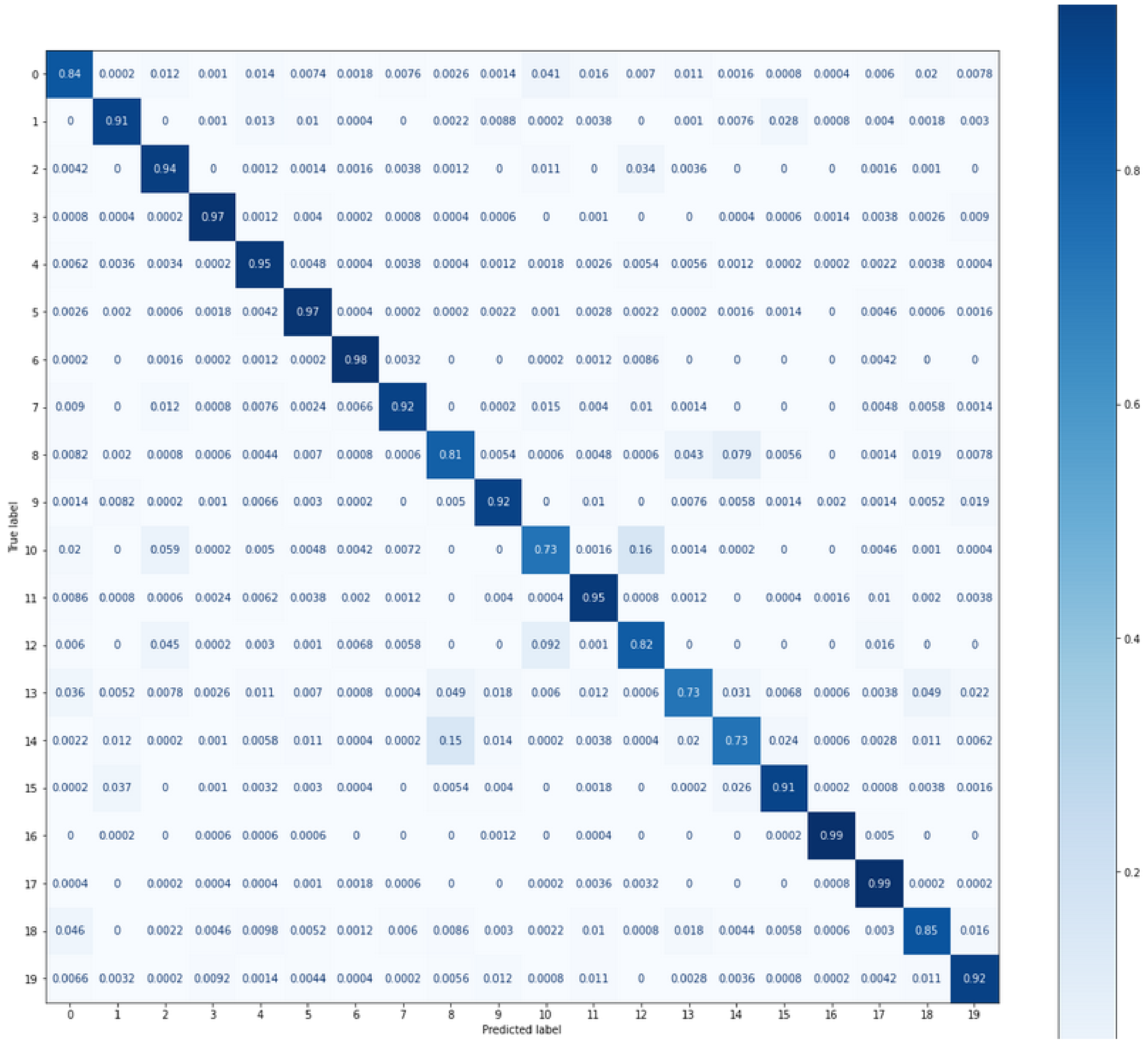


Figure 18: Confusion Matrix for Gradient Boosting on test set



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0.84	0.0002	0.012	0.001	0.014	0.0074	0.0018	0.0076	0.0026	0.0014	0.041	0.016	0.007	0.011	0.0016	0.0008	0.0004	0.006	0.02	0.0078
1	0	0.91	0	0.001	0.013	0.01	0.0004	0	0.0022	0.0088	0.0002	0.0038	0	0.001	0.0076	0.028	0.0008	0.004	0.0018	0.003
2	0.0042	0	0.94	0	0.0012	0.0014	0.0016	0.0038	0.0012	0	0.011	0	0.034	0.0036	0	0	0	0.0016	0.001	0
3	0.0008	0.0004	0.0002	0.97	0.0012	0.004	0.0002	0.0008	0.0004	0.0006	0	0.001	0	0	0.0004	0.0006	0.0014	0.0038	0.0026	0.009
4	0.0062	0.0036	0.0034	0.0002	0.95	0.0048	0.0004	0.0038	0.0004	0.0012	0.0018	0.0026	0.0054	0.0056	0.0012	0.0002	0.0002	0.0022	0.0038	0.0004
5	0.0026	0.002	0.0006	0.0018	0.0042	0.97	0.0004	0.0002	0.0002	0.0022	0.001	0.0028	0.0022	0.0002	0.0016	0.0014	0	0.0046	0.0006	0.0016
6	0.0002	0	0.0016	0.0002	0.0012	0.0002	0.98	0.0032	0	0	0.0002	0.0012	0.0086	0	0	0	0	0.0042	0	0
7	0.009	0	0.012	0.0008	0.0076	0.0024	0.0066	0.92	0	0.0002	0.015	0.004	0.01	0.0014	0	0	0	0.0048	0.0058	0.0014
8	0.0082	0.002	0.0008	0.0006	0.0044	0.007	0.0008	0.0006	0.81	0.0054	0.0006	0.0048	0.0006	0.043	0.079	0.0056	0	0.0014	0.019	0.0078
9	0.0014	0.0082	0.0002	0.001	0.0066	0.003	0.0002	0	0.005	0.92	0	0.01	0	0.0076	0.0058	0.0014	0.002	0.0014	0.0052	0.019
10	0.02	0	0.059	0.0002	0.005	0.0048	0.0042	0.0072	0	0	0.73	0.0016	0.16	0.0014	0.0002	0	0	0.0046	0.001	0.0004
11	0.0086	0.0008	0.0006	0.0024	0.0062	0.0038	0.002	0.0012	0	0.004	0.0004	0.95	0.0008	0.0012	0	0.0004	0.0016	0.01	0.002	0.0038
12	0.006	0	0.045	0.0002	0.003	0.001	0.0068	0.0058	0	0	0.092	0.001	0.82	0	0	0	0	0.016	0	0
13	0.036	0.0052	0.0078	0.0026	0.011	0.007	0.0008	0.0004	0.049	0.018	0.006	0.012	0.0006	0.73	0.031	0.0068	0.0006	0.0038	0.049	0.022
14	0.0022	0.012	0.0002	0.001	0.0058	0.011	0.0004	0.0002	0.15	0.014	0.0002	0.0038	0.0004	0.02	0.73	0.024	0.0006	0.0028	0.011	0.0062
15	0.0002	0.037	0	0.001	0.0032	0.003	0.0004	0	0.0054	0.004	0	0.0018	0	0.0002	0.026	0.91	0.0002	0.0008	0.0038	0.0016
16	0	0.0002	0	0.0006	0.0006	0.0006	0	0	0	0.0012	0	0.0004	0	0	0	0.0002	0.99	0.005	0	0
17	0.0004	0	0.0002	0.0004	0.0004	0.001	0.0018	0.0006	0	0	0.0002	0.0036	0.0032	0	0	0	0.0008	0.99	0.0002	0.0002
18	0.046	0	0.0022	0.0046	0.0098	0.0052	0.0012	0.006	0.0086	0.003	0.0022	0.01	0.0008	0.018	0.0044	0.0058	0.0006	0.003	0.85	0.016
19	0.0066	0.0032	0.0002	0.0092	0.0014	0.0044	0.0004	0.0002	0.0056	0.012	0.0008	0.011	0	0.0028	0.0036	0.0008	0.0002	0.0042	0.011	0.92

Figure 19: Confusion Matrix for Extreme Gradient Boosting on test set