CS 4701 Final Project

Brian Ling (bjl95), Qiaochu Xiong (qx27), Michael Zhou (mgz27)

Neural Networks: When are they Useful? An Analysis of Three Different Datasets

Video: youtube.com/watch?v=rg1PmR6-gYc

Github: https://github.coecis.cornell.edu/mgz27/cs4701-project

Introduction

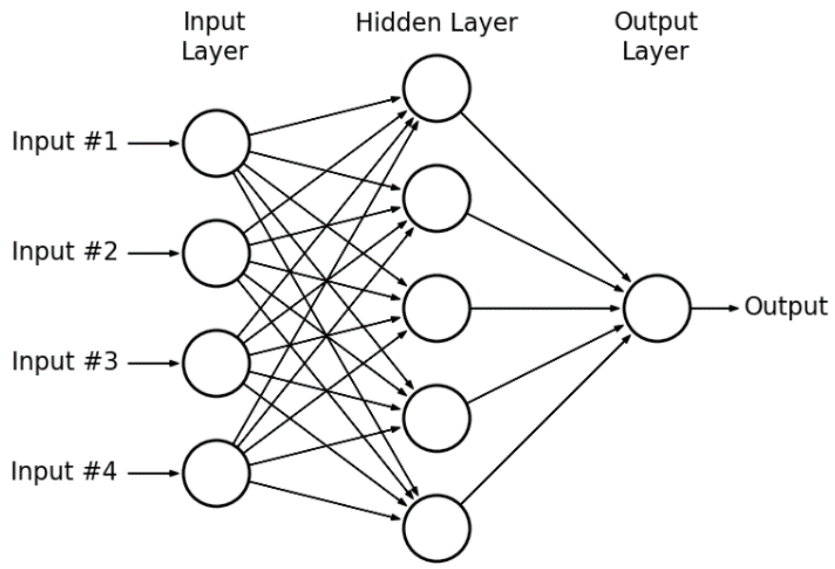
This project illuminates the different types of situations when neural networks are met, as analyzed on different datasets. We built a suite of neural networks - different implementations of fully-connected neural networks (FCNNs) and convolutional neural networks (CNNs). We tested some of these architectures on the following datasets: Musical Notes Datasets, Email Spam Classification Dataset, and the UCI Wine Quality Dataset. We analyzed both the test accuracies and runtimes of each model for each dataset. Our results corroborate that convolutional neural networks (CNNs) work well for image-based datasets (problems including image classification), at the cost of a longer training time due to the complex nature of its architecture. The size of the dataset also contributes to the training time, as a larger dataset implies a longer training time because of a larger input space. Moreover, FCNNs outperform CNNs on datasets that are densely populated by zero. We also noticed that weak correlations between predictors and unbalanced class distributions within the dataset yields a low test accuracy.

Dataset 1: Musical Notes Datasets

The Musical Notes Datasets consist of two different balanced datasets of 5000 musical notes - one for 28 x 28 images (small), the other for 64 x 64 images (large) - each consisting of whole, half, quarter, eighth, and sixteenth notes, 1000 per category **[1]**. The goal is to classify each musical note into each category based on the grayscale pixel values (0-255) of the images. Since all features are inherently at uniform scale, normalization is not needed for this dataset. We also wanted to measure how efficient it was to train the examples and do model selection, along with how the size of the images can affect this. For efficiency, we parallel workers for model selection. We chose to use fully-connected neural networks (FCNNs), which we started off using as a baseline, along with convolutional neural networks (CNNs), which are commonly used in image classification tasks.

Model 1: Fully-Connected Neural Network:

As a baseline model, we first tried classifying the musical notes by implementing a fully connected neural network (FCNN). Also known as a multi-layered perceptron (MLP), an FCNN is composed of a series of fully connected layers that connect every node in one layer to every node in the other layer **[2]**. A fully connected layer is a function from R^m to R^n, where m and n are the number of nodes in their respective layers **[2]**. This means that each output dimension depends on each input dimension, and many different functions, called activation functions, can be applied between the layers. A major advantage of this network is that there are no special assumptions that need to be made about the input, making them broadly applicable **[2, 3]**.



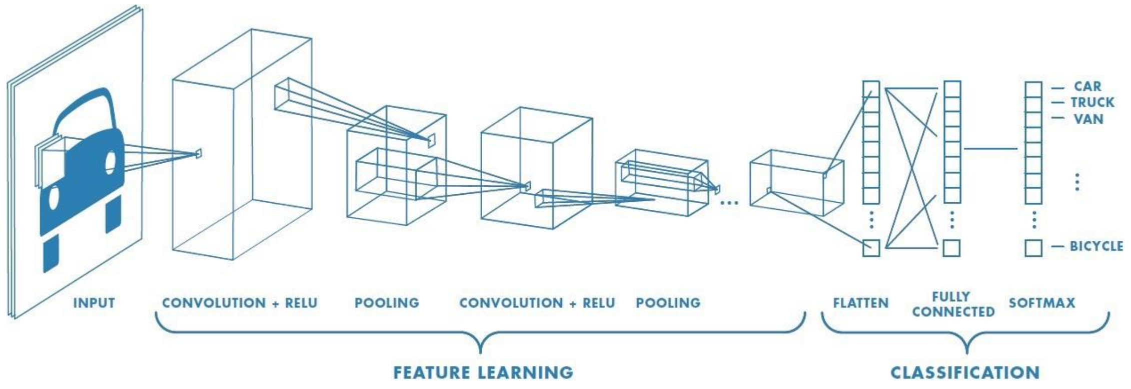
We implemented scikit-learn's neural network library using the default 'adam' solver, as it works well on relatively large datasets in terms of training time and CV score. We first split the data into a 80-20 train-test split (80% training, 20% test data). Because the features only consisted of pixel values, no normalization was needed. In order to choose the best parameters for the neural network, we performed a full grid search over different parameters such as the following: whether or not there is early stopping (True, False), hidden layer sizes (100, 200, 300, 400, 500), activation functions (relu, tanh, logistic), learning rates (constant, invscaling, adaptive), and learning rate initializations (0.0001, 0.001, 0.01, 0.1). When we ran the neural network using the 28x28 image data, the grid search found the following optimized parameters: the activation function was logistic, no early stopping, hidden layer size of 400, learning rate is constant, and the learning rate initialization is 0.0001. This produced training and test accuracies of 0.92225 and 0.88, with CV search and training times of 6134.61 and 37.56 seconds respectively. When we ran the neural network using 64x64 images, a different set of optimized parameters were selected, which are as follows: relu activation with early stopping, hidden layer size 500, adaptive learning rate, and learning rate initialization 0.0001. For this set of parameters, the bigger dataset took 25156.276 seconds for grid search CV. The training took 78.15 seconds. This produced a training accuracy of 0.8815 and a test accuracy of 0.811.

The timing with respect to the FCNN varied based on the parameters used. Splitting the data by activation functions, learning rates, learning rate initialization, and ability to stop early showed interesting overall trends. One key trend with the data pertained to the parameter of early stopping, where with early stopping the mean fit time seemed to be less with data lines of initial learning rate initializations being closer together. Another key difference between time is that tanh had shorter mean fit times on average than both relu and logistic activation functions for both all learning rates and for both options of early stopping. While constant, invscaling, and adaptive learning rates all had different approaches with different CV errors, training accuracy, and testing accuracy, there were no distinct differences between their runtimes.

After implementing this neural network with the adam solver, we wanted to explore how the LBFGS solver would perform on our data set, as it converges faster and performs better for small data sets. Specifically, we wanted to see how much does using the LBFGS solver affect the final test accuracy. We changed the grid search CV to use this solver, and after running for 11 hours on the 28x28 images and observing that it is barely halfway finished, we concluded that the LBFGS is not feasible for our 5000-example large dataset. Therefore, we did not try to use the LBFGS solver for the 64x64 images. Additionally, in an attempt to decrease the amount of time it took to run the network, we tried running a randomized grid search CV over the same parameters. The non-randomized grid search took 6134.61 seconds to run on 28x28 images and 25156 seconds to run on the 64x64 images. The randomized grid search ended up selecting similar optimized parameters for the 28x28 images, producing a training accuracy of 0.89175 and a testing accuracy of 0.844, and taking 539.979 seconds to run. For the 64x64 images, it selected optimized parameters producing similar training and testing accuracies (0.86675 and 0.826 respectively), though it significantly decreased the runtime as it only took 2366.952 seconds to run.

Model 2: Convolutional Neural Network:

Convolutional Neural Networks (CNNs) are particularly useful when working with image data, so CNN is very suitable for the music note image classification dataset. It contains one or many pairs of convolutional and pooling layers **[4]**. After convolutional and pooling layers, a flatten layer follows, and then a dense layer, and finally an activation layer. Since we wanted to classify a new image into one of the five classes: ‘Whole’, ‘Half’, ‘Quarter’, ‘Eight’, and ‘Sixteenth’, we need the dense layer to have output dimension 5. Since we are working on a multi-class classification task, for the activation layer, we use the activation function ‘softmax’, which converts the output of the dense layer into probabilities of an image belonging to each of the five classes **[5]**. Finally, when we compiled our models, we used the ‘categorical_crossentropy’ loss function **[7]**; for metrics, we used ‘accuracy’, which measures how likely predictions we get are the same as the true label **[8]**.



When preprocessing data, we divided our dataset into 80% train-validation, and 20% testing. We further divided the 80% train-validation data into 80% train and 20% validation. We removed any training instances that contain the following values: NAN, infinity, and negative infinity. Due to the required input format of CNN, we reformatted datasets into the form of (number of instances, 28, 28, 1) for 28x28 images, and (number of instances, 64, 64, 1) for 64x64 images. The last dimension, which represents color channels, is 1 for both 28x28 images and 64x64 images because our images are grey **[6]**. We also changed the representation of class labels from strings into categorical variables, since the categorical_crossentropy loss function requires class labels to be represented by one-hot encoding.

When training model for 28x28 images, we manually tuned different combinations of parameters, for instance: number of convolutional/pooling pairs (1,2), dimensions of output for convolutional layers (16, 32, 40), activation functions (relu, sigmoid, tanh), and optimizer (adadelta, adam). For 28 x 28 images, we found that CNN with 2 convolutional/pooling pairs behaves consistently better compared to CNN with 1 convolutional/pooling pair. For CNN models with 2 convolutional/pooling pairs, models with output size (32, 40) for the first and second convolutional layers, respectively, have better performance than models with output size (16, 32) for the first and second convolutional layers, respectively. Also, for activation functions, both ‘tanh’ and ‘relu’ are able to achieve accuracy more than 0.9, and ‘tanh’ behaves better than ‘relu’. However, ‘sigmoid’ has a testing accuracy of 0.7670. For optimizers, ‘adadelta’ is able to achieve accuracy more than 0.9, while performance of ‘adam’ is so bad that the model accuracy doesn’t even reach 0.5. As a result, the best model for 28 x 28 images has two convolutional/pooling pairs, with ‘tanh’ as activation function for both the first and the second convolutional layer, with output dimensions (32, 40), respectively, one flatten layer, one dense layer with parameter 5, and one activation layer with activation function ‘softmax’. When compiling the model, loss is ‘categorical_crossentropy’ and optimizer is ‘adadelta’. This model yielded 0.9636, 0.9350, 0.9340 training, CV, and test accuracies. Training time is 1772.190 seconds. We manually turned the same set of parameters for 64 x 64 images using the same procedure described before, and the same set of values for parameters are chosen. For 64 x 64 images, the training, CV, and test accuracies were 0.9866, 0.9588, and 0.9600 respectively. Training time is 15400.268 seconds.

Model Comparison and Analysis:

Looking at the model performance tables below, the CNN model is far superior compared to the FCNN when it comes to test accuracy. This makes sense, considering CNN models are the most common models used to train superior models, as they contain convolutional layers that encode important image attributes. On the other hand, FCNNs (aka MLPs) do not do as well in image classification tasks such as musical note image classification, since FCNNs only contain flattened vectors, which do not include spatial information **[9]**. However, the training time for CNN is significantly longer than FCNN, as CNN contains 3D layers while FCNNs consist of only flattened vectors, which implies that CNN contains more parameters to train - a higher time complexity. The parameters for the CNN model are tuned manually, as it is difficult to perform cross-validation on this dataset; hence, there is no cross validation for our CNN method. We can also see that the size of the images (i.e. the dataset) also plays a significant role in terms of training time for the CNN model, as it takes 1772 seconds on 28 x 28 images, and a whopping 15400 seconds on 64 x 64 images. The cross-validation search time for the FCNN model is also affected similarly, as CV search takes significantly longer for 64 x 64 images than 28 x 28 images. When there's a larger dataset, there are more parameters you have to keep track of, as more values are inputted into the model, so a larger computation cost is justified for a larger dataset. To sum up in one sentence, CNNs work congenially for image classification tasks, as corroborated by the test accuracies for the Musical Notes Datasets.

Model Performance Tables:

Small Dataset (28 x 28 Images):

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Fully-Connected NN	Grid	0.88	6134.6064739227295	37.561065912246704
Fully-Connected NN	Randomized (20 iterations)	0.844	539.9790909290314	31.42546510696411
CNN	N/A (Manually tuned)	**0.9340000152587891**	N/A (Manually tuned)	1772.1896510124207

Large Dataset (64 x 64 Images):

Algorithm	CV Search Method	Test Accuracy	Cross-validation	Training Time

			Search Time (seconds)	(seconds)
Fully-Connected NN	Grid	0.811	25156.27631998062	78.14718914031982
Fully-Connected NN	Randomized (20 iterations)	0.826	2366.9517533779144	77.53696775436401
CNN	N/A (Manually tuned)	**0.9599999785423279**	N/A (Manually tuned)	15400.268100738525

Dataset 2: Email Spam Classification Dataset

The second dataset we investigated is Email Spam Classification **[18]**. Each row represents a different email. The first column contains email names, and the last column indicates whether the email is spam, with 1 represents spam and 0 represents not spam. The remaining columns are the most common words that occur in all the emails in this dataset, excluding non-alphabetical words. We want to classify whether an email is spam or not based on frequencies of common words in the email.

Compared to Musical Notes Datasets, Email Spam Classification Dataset has a lot more features than the dataset of 28 x 28 images, and a lot less features than the dataset of 64 x 64 images, since the Email Spam Classification dataset has 3000 features, the dataset of 28 x 28 images has 784 features, and the dataset of 64 x 64 images has 4096 features. Data distribution is also very different between Email Spam Classification Dataset and Musical Notes Dataset. Email Spam Classification dataset has a lot of zero entries in addition to entries that are populated with non-zero values. Moreover, these non-zero entries are not distributed uniformly. Both small and large Musical Notes Datasets are densely populated with non-zero values, and their non-zero entries are uniformly distributed, since the majority of them have values around 252.

On the other hand, there are also similarities between the Email Spam Classification Dataset and small and large Musical Notes Datasets. First, both of them have integer entries. Second, their scales of data are similar, since their entries all have values that are less than 1000. As a result, we started with training and testing the performance of two types of neural networks (Fully-Connected Neural Network and Convolutional Neural Network) we used for Musical Notes Datasets to see how resulting neural networks perform on the Email Spam Classification dataset.

Model 1: Fully-Connected Neural Network

We started our investigation from Fully-Connected Neural Network since it is simpler and generalized better than Convolutional Neural Network. As for Musical Notes Datasets, we implemented the neural network from scikit-learn library. We splitted data into 80% for training and 20% for testing. In the data preprocessing step, we removed all rows that have values none, infinity, and negative infinity. We didn't need to normalize values in the dataset because they are all counts of frequencies in the same scale. We performed grid search to choose parameters. We examined whether or not we want to include early stopping (True, False). Since there are 3000 features, it is possible that the resulting model will overfit if we used all features to train the neural network. But at the same time, we didn't want to lose accuracy due to truncating available data inappropriately. For hidden layer sizes, we chose either 100 or 150. Since there are 3000 features, we didn't choose small hidden layer sizes because they can't take into account the complexity of the dataset. For the activation function, we chose either "relu" or "tanh", since these two are most commonly used activation functions **[13]**. For the initial learning rate, we chose either 0.01 or 0.1. Starting with either 0.01 or 0.1 as learning rate, we checked whether we need to adapt learning rate to our specific problem or not to gain good performance by setting learning rate to either constant or adaptive. We used the Adam solver. After performing 5 fold cross-validation for each of 32 different parameter combinations, optimized parameter combination grid search chose is relu for activation function, True for early stopping, 100 for hidden layer size, constant for learning rate, and 0.01 for initial learning rate. Training time for the fully-connected neural network using optimized parameter combination is 10.122442722320557 seconds. Training accuracy for this model is 0.9973410684070583, and testing accuracy on the 20% testing data is 0.9729468599033816, which is a very accurate model.

Also using 80-20 train-test split, we performed randomized search to select parameters from the same set of parameter values as described above. We did 5 fold cross validation on each of 32 parameter combinations, and the following optimized parameter combination was chosen: 0.01 for initial learning rate, adaptive for learning rate, 150 for hidden layer size, False for early stopping, and relu for activation function. Training time for the fully-connected neural network using this set of optimized parameters is 47.68662929534912 seconds. Training accuracy for this model is 1.0, and testing accuracy on the 20% testing data is 0.9797101449275363. This model is even more accurate compared to the model grid search chose.

Model 2: Convolutional Neural Network

The second neural network we used is Convolutional Neural Network. It is more complicated than Fully-Connected Neural Networks. It is also one of the recommended neural networks for identifying spam emails using email subject lines **[14]**, and it is more powerful than Recurrent Neural Networks **[15]**. Thus, we wanted to know whether we would be able to get a more accurate model.

We took out the last column as true labels, and all columns except for the first and the last column as training, validation, and testing data. In consistency with fully-connected neural networks, we divided the entire dataset into 80% for training and 20% for testing. We further divided the training data into 80%

for training and 20% for validation. For data preprocessing, we removed all rows that have values of none, infinity or negative infinity. We reshaped training, validation, and testing data from rows to columns to satisfy the input data requirements of Convolutional Neural Networks. We also stored true labels into python lists, and then converted true labels into categorical data with two categories. In small and large Musical Notes datasets, each row represents pixel values for an image, so we reshaped the data into two-dimensional squares representing images, and trained Convolutional Neural Networks with one or more two-dimensional convolutional layers. But in Email Spam Classification Dataset, each entry that is not in the first and last column represents number of occurrences of a distinct word in an email, so instead of reshaping dataset into two-dimensional squares, we trained Convolutional Neural Networks with one or more one-dimensional convolutional layers.

We started our model training using only one one-dimensional convolutional layer. Thus, the model consists of one one-dimensional convolutional layer with input shape (3000, 1), followed by a one-dimensional max pooling layer with pooling size equals to 2, followed by a flatten layer, followed by a dense layer with output dimension 2 since the Email Spam Classification Dataset has two categories, followed by an Activation layer, followed by compilation of the model with binary_crossentropy as loss function and accuracy as metrics. Binary_crossentropy penalizes the probabilities based on their distance to the corresponding true binary labels so binary_crossentropy is suitable for our email spam classification problem **[16]**. When tuning parameters, we used 16, 32, and 64 as convolutional layer output dimensions, since 32 is used as convolutional layer output dimension in email spam classification using subject line **[14]**. Similar to fully-connected neural networks, we used relu, tanh as the convolutional layer activation function. For output layer activation functions, we used softmax and sigmoid since they are commonly used output layer activation functions for classification problems **[13]**. We used RMSprop, AdaGrad, and Adam, for optimizer when compiling the model. RMSprop is used in Email Spam classification based on subject line, so we initiated our investigation from RMSprop **[14]**. We tested AdaGrad since it can be useful when the most important information is sparse, and nonzero entries, which are critical for determining characteristics of spam emails, in the Email Spam Classification dataset is sparse **[17]**. Adam combines advantages of RMSprop and AdaGrad, and it provided accurate predictions for fully-connected neural networks, so it would probably generate accurate models for Convolutional Neural Networks **[17]**. We didn’t use SGD optimizer since it is rarely used **[17]**. Even if it provides accurate predictions for our email spam classification problem, the resulting model is less likely to be generalized to other problems. Models with 16 as convolutional layer output size achieve maximum accuracy after training for 22 epochs, which is faster than models with 32 as convolutional layer output size, which achieve maximum accuracy after training for 30 epochs. Accuracies for models with 32 as convolutional layer output size are similar to accuracies for models with 16 as convolutional layer output size. Both “relu” and “tanh” as convolutional layer activation functions also have similar performance in terms of accuracy and running time. With slightly better accuracy, models with 32 as convolutional layer output size and “relu” as convolutional layer activation function have the best performance, with an accuracy of 0.9787439703941345. After changing the optimizer of the model compilation step from RMSprop to AdaGrad, accuracy decreases to 0.947826087474823. Accuracy increases to 0.9758453965187073 after changing the activation function of the output layer from Adagrad to Adam. Since RMSprop gives the best accuracy, we changed the optimizer back to RMSprop, and modified the output layer activation function from softmax to sigmoid. The behaviour of the model is quite steady, giving an accuracy of 0.9748792052268982 after being trained for 30 epochs. We also tested models with 64 as convolutional layer output size, but we discovered that accuracy is consistent with what we get with smaller convolutional layer output size, but training time is increased significantly, by approximately 100 seconds, so we didn’t do further exploration.

We tuned our Convolutional Neural Networks with two one-dimensional convolutional layers using the same set of parameter values. For the additional convolutional layer, we also used 16, 32, and 64 as its output dimensions. With output dimension (16, 32), which means that the first convolution layer has output dimension 16 and the second convolution layer has output dimension 32, testing accuracy is lower than accuracy of the best-performing model with one one-dimensional convolution layer. This also happened to models with output dimension (32, 64). With output dimension (32, 40), testing accuracy is above testing accuracies with output dimension (16, 32) and (32, 64). Using tanh as the activation function for convolutional layers, changing the optimizer to Adam, and changing output layer activation function to sigmoid all give worse accuracies compared to the accuracy of best-performing model with one one-dimensional convolution layer.

Model Comparison and Analysis:

Among Convolutional Neural Networks, the best accuracy is achieved by the model with one one-dimensional convolution layer with relu as activation function and output size 32, with softmax as output layer activation function and RMSprop as the optimizer. However, it is less accurate than the fully-connected neural network selected by randomized search, which is the most accurate model. This is reasonable since most entries of Email Spam Classification are zero, and Convolutional Neural Network is probably too complicated for this dataset. This can also be seen from the fact that one-dimensional convolutional neural networks have similar or even better performance compared to two-dimensional neural networks. Moreover, since the Email Spam Classification Dataset is densely populated with zero, our result shows that fully-connected neural networks work better for datasets which have zero as values for a significant proportion of their entries. Although both Email Spam Classification Dataset and Musical

Notes Datasets consist of numerical values, values of entries and distribution of entry values across the entire dataset cause significant difference in performance for both fully-connected neural networks and convolutional neural networks.

Model Performance Table:

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Fully-Connected NN	Grid	0.9729468599033816	1817.4925224781036	10.122442722320557
Fully-Connected NN	Randomized	**0.9797101449275363**	2056.2531270980835	47.68662929534912
CNN	N/A (Manually tuned)	0.9787439703941345	N/A (Manually tuned)	202.64555978775024

Dataset 3: UCI Wine Quality Dataset

We used the Wine Quality Dataset from the UCI Machine Learning Repository as our dataset for analysis **[10]**. It contains two datasets, each consisting of red wines and white wines. Each dataset contains 12 variables: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality (integers ranging from 0 to 10). The goal is to predict the wine quality. There are 1599 red wine samples and 4898 white wine samples. We chose to use two implementations of fully-connected neural networks - one with a single hidden layer and another with multiple hidden layers - since our dataset features consist of continuous values.

Class Distribution and Correlations Among Predictors

We plotted the class label distributions using R code for red wines and white wines separately, as displayed in figure 3.1. Note that R is only used for plotting the class distribution and inter-predictor correlations, but not for fitting any of our models for any dataset. We can see that the majority of labels are either a 5, 6, or 7, with very few labels outside this range. The lowest quality given in both datasets is a 3. We see a few white wine samples with a quality of 9, while the maximum quality in all red wine examples is 8. Looking at this plot, this dataset is highly imbalanced.

We then plotted the correlations between each predictor for the red (left) and white (right) wine datasets, in the form of heat maps (figure 3.2) and histograms (figure 3.3). We notice that most of the predictor pairs in the heat map not on the main diagonal have a light orange color, which denotes a weak inter-predictor relationship. We also see that most of these predictor pairs have a correlation closer to 0, which similarly denotes very little correlation among predictors. We can see later that this weak correlation among predictors, as well as the highly imbalanced class distribution, makes our models very difficult to yield high performances. Very importantly, we can see that the red wine dataset has more variable pairs with higher correlations, which makes it easier for models to pick up patterns within predictors; as we see later, this contributes to models predicting more accurately for red wines than white wines.

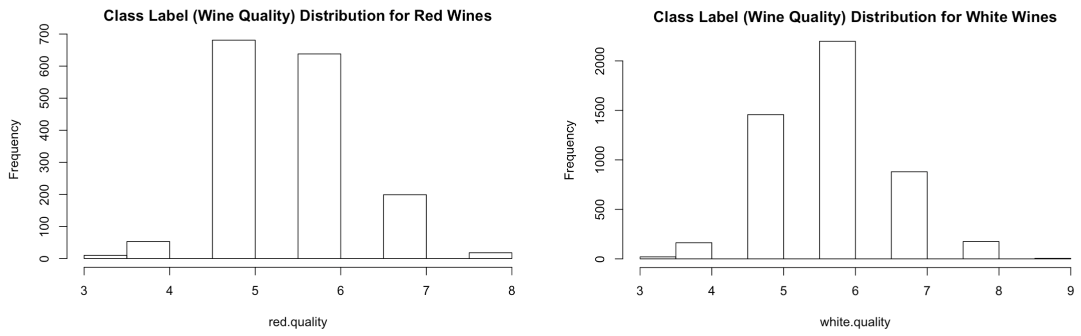


Figure 3.1: Class label distributions for red wines (left) and white wines (right)

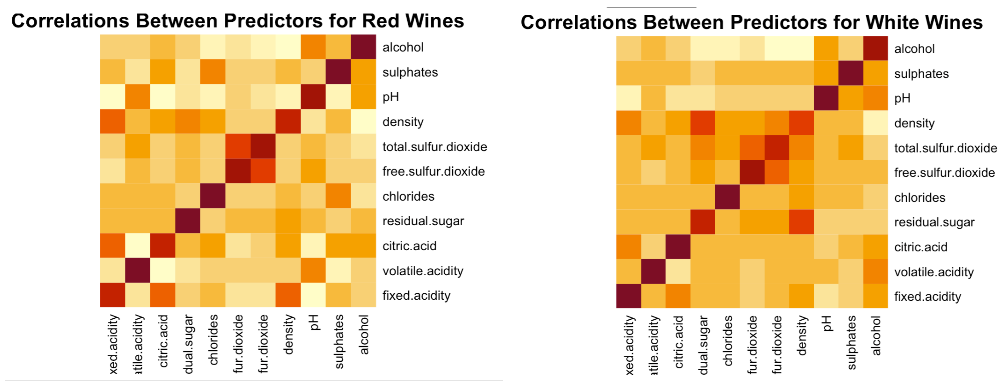
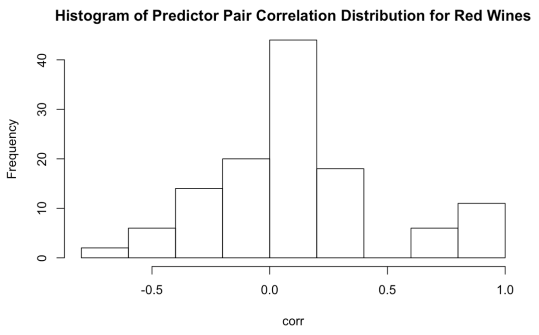


Figure 3.2: Heat maps showing correlations between predictors for red wines (left) and white wines (right). Lighter color denotes weaker relationship between two predictors.



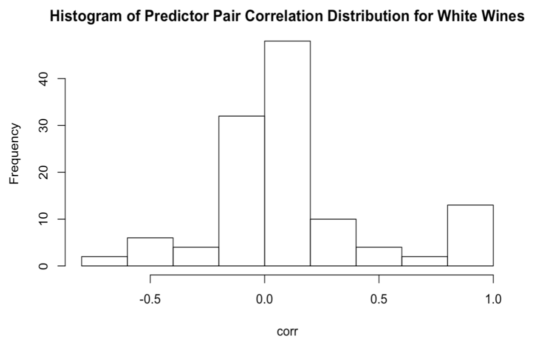


Figure 3.3: Histograms showing correlations between predictors for red wines (left) and white wines (right).

Model 1: Single Hidden Layer Fully-Connected Neural Network

For this model, we use the exact same methods (grid search and random search) as described in model 1 of dataset 1, except that we set the random seed for the MLPClassifier (from Scikit-learn) to 42. We ran this model for the red and white wines datasets separately. Since our class labels do not start at 0, we converted them this way by using a LabelEncoder. Since we do have non-uniform features for this dataset, we normalized our features with a StandardScaler (using default configurations). Since our implementation in section 1 of dataset 1 is a single hidden layer model, we call this the single hidden layer FCNN model.

We first ran our randomized search CV. Using the exact same parameter configurations as in model 1 of dataset 1, after 151.85 seconds of randomized CV search (20 random configurations), we obtain for the red wines dataset the optimal hyperparameter grid of learning_rate_init = 0.001, adaptive learning rate, hidden layer size 300, no early stopping, and relu activation, as this configuration yields the highest 5-fold CV validation score; we then write the CV results to a txt file; finally, after retraining using the optimal configurations on the entire training set (which took about 15.6 seconds), we obtain a 0.6375 test accuracy. For the white dataset, we obtain the optimal hyperparameters of learning_rate_init = 0.001, adaptive learning rate, hidden layer size 500, no early stopping, and relu activation, after 688.173 seconds of random CV search; we then write the CV results to a txt file; finally, after retraining using the optimal configurations on the entire training set (which took about 46.4 seconds), we obtain a 0.6183673469387755 test accuracy.

We then ran our grid search CV. Using the exact same parameter configurations as in model 1 of dataset 1, after 2361.74 seconds of grid search, we obtain for the red wines dataset the optimal hyperparameter grid of learning_rate_init = 0.1, constant learning rate, hidden layer size 500, no early stopping, and logistic activation, as this configuration yields the highest 5-fold CV validation score; we then write the CV results to a txt file; finally, after retraining using the optimal configurations on the entire training set (which took about 3.13 seconds), we obtain a 0.653125 test accuracy. For the white dataset, we obtain the optimal hyperparameters of learning_rate_init = 0.001, constant learning rate, hidden layer size 500, no early stopping, and relu activation, after 17003 seconds of grid CV search; we then write the CV results to a txt file; finally, after retraining using the optimal configurations on the entire training set (which took about 47 seconds), we obtain a 0.6479591836734694 test accuracy.

So as we can see, when we switch from random to grid search CV, the test accuracy increased from 0.6375 to 0.653125 for the red wines subset, and 0.61837 to 0.648 for the white wines subset. But because we are looping through the entire set of 360 hyperparameter configurations, grid search takes hours to run while randomized search only takes minutes. Yet we do see a boost in test performance, as the random search with 20 iterations did not find the actual optimal parameters, while grid search is guaranteed an optimal solution.

Model 2: Multi Hidden Layer Fully-Connected Neural Network

We then tried a new implementation of the fully connected neural network - using multiple hidden layers and using Keras. We chose to use Keras because the MLPClassifier from Scikit-learn does not support more than one hidden layer, while Keras does. We once again converted all class labels to start from 0 using a LabelEncoder, but this time, we chose to convert them further into a one-hot encoding, as is the case with most neural network implementations for multiclass classification. Once again, we normalized all features using a StandardScaler (and default parameters). Like all other models, we used a 80-20 train-test split.

Our model implementation is as follows. For our Sequential network, we first add a dense layer with the kernel initialization set as ‘he_uniform’, and set our number of hidden units to 50, 100, 150, …, 400 for our hyperparameter grid. We then add a dropout layer with 0.2 dropout rate. We then add 1 to 12 more dense layers with the number of units ranging from 50, 100, …, 400, and then another dropout layer with the dropout rate ranging from 0, 0.1, …, 0.5. We add one more of these layer combinations with the same thing. Our output layer with softmax activation for the red dataset consists of 6 units, while the white dataset consists of 7 units, since there are 6 and 7 different class values for the red and white datasets respectively. We compile our model using the adam optimizer with MSE as the loss and metrics. For both subsets, we use a Keras tuner to run 60 iterations of random search (1 execution per trial, with the MSE as objective) **[12]**. Our tuners use 100 epochs per trial, with our early stopping criterion as no improvement in validation loss after 10 iterations. Our best model is the model that yields the lowest validation loss. We saved our best models into h5 files so that we can run them later if necessary. Our Keras tuner is saved automatically, so that we can run them without having to rerun the entire CV trial (unless we set overwrite=True for our random search tuner).

On the red subset, it took 6 minutes and 16 seconds for the random search CV, yielding 52.15% and 52.5% train and test accuracies respectively. On the white subset, it took 13 minutes and 24 seconds for the random search CV, yielding a pitiful 26% and 25% train and test accuracies respectively. Since the Keras tuner automatically saves the best model, we did not need to retrain this model on the entire training set as in the single-hidden-layer model.

Model Comparison and Analysis:

We can clearly see that the MLPClassifier (i.e. single-hidden-layer FCNN) yields a higher test accuracy, as there is a dichotomy between the test scores - the single-hidden-layer FCNN (using MLPClassifier) produces test accuracies in the 60s, while the multi-hidden-layer (using Keras and Dense layers) yields a test accuracies in the 20s and 50s (for white and red respectively). We could hypothesize that this low accuracy in the latter model could either be caused by the loss function choice, the dropout rates between layers (which we set from 0.1 to 0.5 in the CV search), or by the early stopping criterion (if the validation categorical cross-entropy loss does not decrease in 10 epochs), since we see that our training accuracies are 0.5215 and 0.26646 for red and white wines respectively (severe underfitting). But since we did not record the different CV results for this model (at least we could not find a way to do so), we cannot tell for sure whether any of these is the case. We have tried using the categorical cross-entropy as our loss and CV evaluation metric, but doing so yielded test accuracies in the 30s and 40s, which are not much better. Since we are using the same core architecture (an FCNN), the average fit times for each of the models are comparable. For both models, we can see that a longer CV search time is needed for white subsets than for red subsets, as there are close to 4 times more white wine examples than red wine examples (4898 white, 1599 red).

Why is our Test Accuracy So Low?

Both our neural network implementations produced an extremely low test accuracy, with as the highest accuracies for the red and white datasets yielded at just over 64%. To see why this is, let's revisit the correlation plots between the predictors described before. As shown in the heat maps and correlation histograms, there is very little correlation between the different predictors, so it is very difficult to find patterns among the variables, because there are not that many predictors (only 11 of them) and they are not very strong indicators. This contributes to low accuracies across different models, since we can't utilize any inter-predictor patterns to better learn about the data **[11]**. Moreover, we can also see that the class labels (wine qualities) are extremely unbalanced, as the bulk of the labels are 5, 6, or 7, yet there are very few examples that are 4 or lower or 8 or above. The models, in turn, will also spit out class labels that are equally unbalanced, as the predictors will also predict mostly 5s, 6s, or 7s, while very rarely predicting any other quality value. In short, the low test performance is mostly ascribed to the inter-predictor correlations and unbalanced class distribution of the datasets themselves, but not the models used for training.

Model Performance Table or Graphs:

Red Wines Dataset:

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Single Hidden Layer Fully-Connected NN	Grid	**0.653125**	2361.737888097763	3.13327693939209
Single Hidden Layer Fully-Connected NN	Randomized (20 iterations)	0.6375	151.85131907463074	15.565123796463013
Multiple Hidden Layer Fully-Connected NN	Randomized (60 iterations)	0.525	376.85603404045105	N/A

White Wines Dataset:

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Single Hidden Layer Fully-Connected NN	Grid	**0.6479591836734694**	17003.405514001846	46.954169034957886
Single Hidden Layer Fully-Connected NN	Randomized (20 iterations)	0.6183673469387755	688.1731026172638	46.393786907196045
Multiple Hidden Layer Fully-Connected NN	Randomized (60 iterations)	0.25204081632653064	804.2011730670929	N/A

<u>Conclusion</u>

We have analyzed three different tasks from three different dataset - Music Notes Classification, Email Spam Classification, and the UCI Wine Quality Datasets. We used the single-layer FCNN models and CNNs for Music Notes Classification, single-layer FCNN models and one-dimensional CNNs for

Email Spam Classification Dataset, along with single and multi-hidden-layer FCNNs for the UCI Wine Quality Datasets. Our results show that the CNNs work very well for image classification tasks such as music notes classification (despite taking longer to train because of more parameters in 3D convolutional layers), single-layer FCNNs has higher testing accuracy comparing to CNNs on Email Spam Classification Dataset, and single-hidden-layer MLP classifiers outperform the multi-hidden-layer FCNN model (mostly due to some bug) for the UCI Wine Quality Dataset. It is also obvious that the size of the dataset can affect the runtimes of the models, as larger datasets naturally yield slower training times. As shown in the third dataset aforementioned, low accuracies can be produced because of lack of inter-predictor correlations (as well as an imbalanced class distribution) within the dataset itself.

Bibliography

1. Musical Notes Datasets:
   https://www.kaggle.com/kishanj/music-notes-datasets
2. Neural Network explanation:
   https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
3. Fully Connected Deep Neural Networks:
   https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html
4. Convolutional Neural Network (CNN):
   https://www.tensorflow.org/tutorials/images/classification
5. Softmax activation function explanation:
   https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=The%20softmax%20function%20will%20output,%5B0%2C%201%2C%200%5D
6. Convolutional Neural Network explanation:
   https://www.youtube.com/watch?v=n2MxgXtSMBw
7. Keras Losses:
   https://keras.io/api/losses/
8. Keras Metrics:
   https://keras.io/api/metrics/
9. CNN vs MLP for Image Classification:
   https://medium.com/analytics-vidhya/cnn-convolutional-neural-network-8d0a292b4498
10. Wine Quality Data Set
   https://archive.ics.uci.edu/ml/datasets/wine+quality

11. Estimating Wine Quality with Machine Learning (AI), 72% Accuracy
https://ai.plainenglish.io/estimating-wine-quality-with-machine-learning-ai-72-accuracy-8a5ff0bab3b2
12. Introduction to the Keras Tuner
https://www.tensorflow.org/tutorials/keras/keras_tuner
13. How to Choose an Activation Function for Deep Learning
https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
14. Detecting a simple neural network architecture for email classification
https://towardsdatascience.com/detecting-a-simple-neural-network-architecture-using-nlp-for-email-classification-f8e9e98742a7
15. TensorFlow - CNN And RNN Difference
https://www.tutorialspoint.com/tensorflow/tensorflow_cnn_and_rnn_difference.htm
16. Binary Cross Entropy/Log Loss for Binary Classification
https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/
17. Guidelines for selecting an optimizer for training neural networks
https://datascience.stackexchange.com/questions/10523/guidelines-for-selecting-an-optimizer-for-training-neural-networks
18. Email Spam Classification Dataset
https://www.kaggle.com/balaka18/email-spam-classification-dataset-csv?select=emails.csv