Deep vs shallow learning: the evaluation of model predictability.

Anitha Banda, Gauri Bhiwgade, Leonard Phillips

Massey University

# Author Details

Anitha Banda - 18046853 - [anithabanda6@gmail.com](mailto:anithabanda6@gmail.com)

Gauri Bhiwgade - 20013346 - [gauri.bhiwgade@gmail.com](mailto:gauri.bhiwgade@gmail.com)

Leonard Phillips - 15232331 - leonardjohnphillips@gmail.com

**Abstract**

The hallmark of a great machine learning practitioner is their ability to apply many different models to many different data sets and produce highly accurate consistent results. In this report that is exactly what we have tried to do. Our goal was to train two shallow learning and two deep learning models on 6 uniquely different data sets and compare the results. The two shallow learning models we have selected for evaluation are K-Nearest Neighbors (KNN) and Light Gradient Boosted Model (LGBM). The two deep learning models we have selected for evaluation are a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN). We discovered that both the LGBM and CNN models performed the best classification accuracy out of the 4 models. Each scored an accuracy of 82.8% and 82.4% respectively.

**Deep Learning Technique - Multi-layer Perceptron.**

A Multilayer Perceptron (MLP) is an artificial neural network that contains multiple layers of neurons (nodes) that are fully-connected to one another. Input data is mapped to a set of output nodes in a directional, feed-forward manner. Each node inside the multilayer perceptron alters the value of the input data slightly, also known as an activation function. There are many types of activation functions that achieve different transformations. Between these activation nodes are weighted edges which determine the effect each input data point has in the activation function.

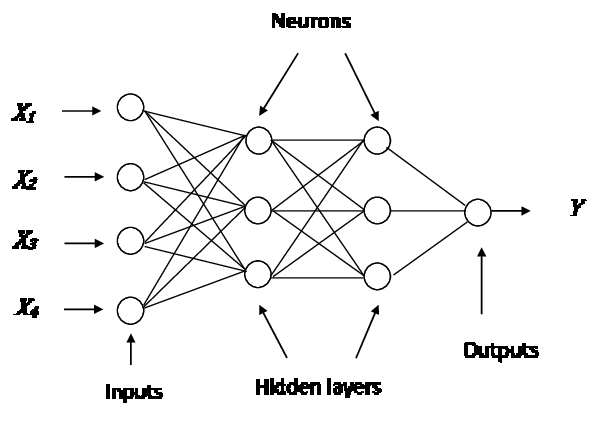


Figure 1. Example of a MLP

Once the input data has pasted through the output layer, it is measured against a ground truth. The distance between the predicted value and ground truth (loss) is then used to update the weighted edge during the back-propagation process. (Werbos 1990). Back-propagation allows the model to finely tune each weight with the goal of reducing the overall loss of the model.

MLP models operate extremely well noisy data such as images. An image containing our target object may also contain various shapes and features we are not interested in. A deep learning model such as MLP works well in identifying patterns in the shapes that give meaning towards our target while ignoring the shapes that do not.

**Convolutional Neural Network**

Common definition of deep learning or Deep Neural Network is nothing but multiple hidden layers stacked upon each other and most of the time it is referred as Artificial Neural Networks (ANN). ANN has become very popular for handling a very large amount of data. ANN consists of 3 layers – Input, Hidden, and Output. But there are some challenges with ANN while solving an image classification problem. The first step for image classification is to convert a 2-dimensional image into a 1-dimensional vector prior to training model which leads to some drawbacks:

1.The number of trainable parameters increases drastically with an increase in the size of image. For example, if the size of image is 224\*224, then the number of trainable parameters at the first hidden layer with just 4 neurons is 602,112 which is huge.

2.ANN loses the spatial features of image. ANN unable to capture sequential information in the input data which is required for dealing with sequence data.

These problems are overcome with the use of Convolutional Neural Network (CNN). Reducing the number of parameters in ANN is the most beneficial aspect of CNN. The important difference between ANN and CNN is that the layers within the CNN are composed of neurons organized into three spatial dimensions of the input, i.e. height, width and the depth. The depth does not refer to the total number of layers within the ANN, but the third dimension of an activation of volume. Unlike standard ANNs, the neurons within any given layer will only connect to a small region of the layer preceding it.

CNN has an excellent performance in machine learning problems. Applications that deal with image data, computer vision, and in natural language processing (NLP) show very amazing results.

Architecture of CNN:

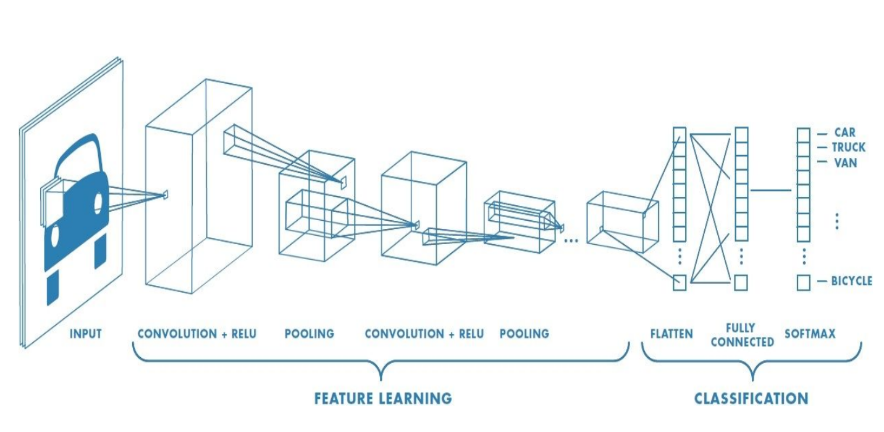
CNN has multiple layers which include convolutional layer, non-linearity layer, pooling layer and fully-connected layer. Parameters are part of the convolutional and fully connected layers and not of pooling and non-linearity layers.

Figure1: Sample Architecture

The Four key areas are:

1. As found in other forms of ANN, the input layer will hold the pixel values of the image.
2. **The Convolutional Layer:**

The layers parameters focus around the use of learnable Kernels or series of filters which aim at extracting local features from the input, and each kernel is used to calculate a feature map or kernel map. In the first layer meaningful features are extracted like edges, corners, texture and lines. Likewise, each layer extracts some important feature but highest-level feature extracted from the last convolution layer. Convolutional layers are also able to reduce the complexity of the model through the optimization of its output through three hyperparameters, the depth, the stride and setting zero padding.

**Depth:** The depth of the volume produced by the convolutional layers can be manually set through the number of neurons within the layer to a the same region of the input.

**Stride:** Kernel size is nothing but the filter size, which revolves around the feature map and the amount by which the filter slides is the stride. It controls how the filter convolves around the feature map.

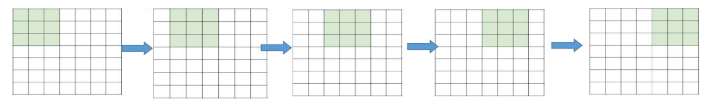


Figure 2: Input size N=7 and Filter Size F = 3 and stride 1 (Output will be 5x5 from input image 7x7) [1]

**Zero Padding:** During the convolution process the loss of information on the border of image might happen. Because they are only captured when the filter slides, they never have the chance to be seen. To resolve this issue, zero padding is used. In the above example the output 5x5 shrinks from 7x7 but by adding one zero-padding, the output will be 7x7, which is exactly the same as the original input.

**Batch Normalization:** Mostly inserted before the activation layer. It makes the network robust to bad initialization of weights. It reduces covariance shift by normalizing and scaling inputs. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

**3 Nonlinearity:**

After the convolution the next layer is non-linearity. It is used to adjust or cut-off the generated output. The following figure shows common types of nonlinearity i.e. sigmoid, tanh, ReLU (Rectified Linear Unit) and softplus.

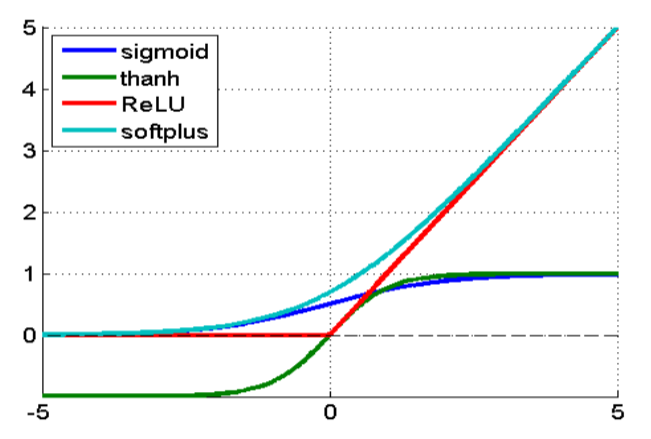


Figure3: Common type of nonlinearity[4]

Following are the main reasons why ReLU is used more often.

1. The saturated function such as sigmoid and tanh cause problems in back propagation, called vanishing gradient. As the neural network design is deeper, the gradient signal begins to vanish, this happens since the gradient of those functions is very close to zero almost everywhere but the center. However, the ReLU has a constant gradient for the positive input.
2. The ReLU creates a sparser representation because the zero in the gradient leads to obtaining a complete zero.

**Pooling**

Pooling is down sampling used to reduce the dimensionality of the representation, and to reduce the number of parameters and computational complexity of the model. Pooling does not affect the number of filters. Max-pooling is one of the most common types of pooling methods. It partitions the image to sub-region rectangles, and it only returns the maximum value (In case of average pooling it returns average value) of the inside of that sub-region. Most common size used in max pooling is 2x2. To avoid down-sampling, stride 1 can be used, which is not common. [2]

**Flattening:** It is converting the data into a 1-dimensional array for inputting it to the next layer. It has been done to create a single long feature vector and it is connected to the final classification model, which is called fully connected layer.

**4 Fully Connected Layer:**

It contains neurons of which are directly connected to the neurons in the two adjacent layers, without being connected to any layers within them.

The major drawback of a fully-connected layer, is that it includes a lot of parameters that need complex computational in training examples. During training, randomly ignore activations by probability p and during testing, use all activations but scale them by p. Effectively prevent overfitting by reducing correlation between neurons.

**Softmax:** A special kind of activation layer, usually at the end of fully connected layer outputs. It produces a discrete probability distribution vector and it is very convenient when combined with cross-entropy loss (works well for classification, e.g., image classification). [3]

**Shallow Learning Technique: k - Nearest Neighbor (k-NN)**

the result, and a large number of neighbors make it computationally expensive in terms of time and memory. k-NN is a case-based learning method, which keeps all the training data for classification. k-NN is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition, image processing and many more. k-NN can be used as a classifier or regressor.

The following two properties can define k-NN well:

Lazy learning algorithm- It is a lazy learning algorithm because it does not have a specialized training phase and uses all data for training while classification.[5]

Non-parametric learning algorithm- It is a non-parametric algorithm because it doesn’t assume anything about the underlying data.

It tries to classify an unknown sample based on the known classification of its neighbors. In k-NN, the relation of a query to a neighboring sample is basically measured by a similarity metric, such as Euclidean distance (or Manhattan, or Hamming distance). However, to apply k-NN we need to choose an appropriate value for k, the whole success of classification is based on different values of k and selected on the basis of best performance.

To improve its efficiency, find some representatives to represent the whole training dataset and use this model for classification i.e. building an inductive learning model from the training dataset and using this model for it. In case of a small number of neighbors, the noise will have a higher influence on the result, and a large number of neighbors make it computationally expensive in terms of time and memory.

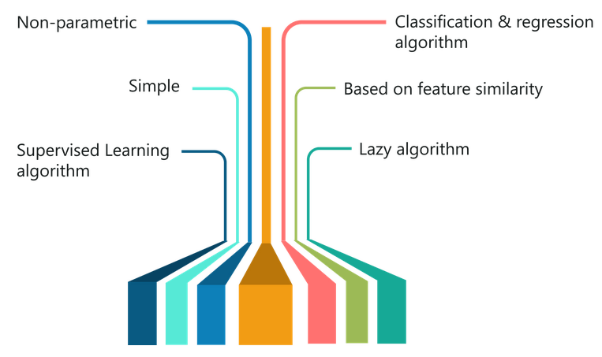


Figure 4: Features of k-NN Algorithm [1]

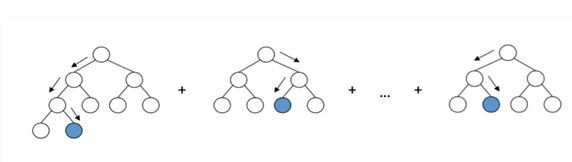
**Data Sets**

The data set selected from this experiment were Forest Cover Type, Epileptic Seizure Recognition.

**LightGBM**:

LightGBM uses a Gradient Boosting framework. LGBM is much faster, uses less memory, handles large and sparse data effectively and provides better accuracy. LGBM accelerates the training process by over 20% by achieving the same accuracy when compared to the other boosting algorithms.

Boosting is an ensemble technique of making a strong learner from the n week learners. Gradient Boosting is a tree-based algorithm, each new decision tree builds on a weighted version of the original dataset where the weights are updated by learning from the previous tree. The algorithm initially assigns equal weights to all the data points. After evaluating each tree/build, it increases the weights of misclassified samples and reduces the weights of correctly classified samples so the next tree/build will focus on the more difficult samples. This process is repeated for a specified number of iterations. Each new tree build will focus on the incorrectly classified samples by the previous tree. Therefore, the final predictions are a weighted average of all the predictions.



Most boosting algorithms like GBM use *Pre-sorted algorithms* which are very efficient and time consuming. A major reason is that for each feature, it needs to scan all the data points to calculate the information gain for all the possible splits. LightGBM uses *Histogram-based algorithms* which buckets continuous features into discrete bins to construct feature histograms.

LGBM uses two techniques to reduce the complexity of the histogram building:

***Gradient-based One-Side Sampling* (GOSS)**

GOSS is a novel sampling method. It only uses the samples with high gradient to calculate the information gain and excludes the samples with low gradient as they are trained well.

***Exclusive Feature Bundling* (EFB)**

EFB is a feature bundling technique.When the data is sparse or has a higher number of features, many features are mutually exclusive, i.e they never take zero values simultaneously. This feature bundling technique finds the exclusive features and carefully bundles into a single feature.

**Epileptic Seizure Recognition**

The Epileptic Seizure Recognition dataset is a re-structured time-series collection of brain read readings in 500 humans. The research captured 4097 data points over a 23.5 second period for each patient. Each data point is an Electroencephalograph (EEG) reading. The dataset was then divided into points per second (4097/23) and then multiplied by the 500 patients sampled. The final dataset totals 11500 rows and 180 columns. The dataset, once loaded into memory, was 15.8MB but after memory reduction, it was reduced to 4MB. This is a 75% reduction in size.

The predictor variables are numeric values between -1885 and 2047 with a mean of -7.66. The target variable is a categorical variable that consists of 5 classes:

1. Recording of seizure activity
2. They recorder the EEG from the area where the tumor was located
3. Yes they identify where the region of the tumor was in the brain and recording the EEG activity from the healthy brain area
4. Eyes closed, means when they were recording the EEG signal the patient had their eyes closed
5. Eyes open, means when they were recording the EEG signal of the brain the patient had their eyes open

As classes 2 through 5 did not record any seizure activity at all, they can be consolidated into one class: non-seizure readings. This transforms a more complex multi-class classification problem into a simplified binary classification. As our goal is to predict epileptic seizures from our data, this is a valid transformation. After the consolidation, there are 9200 non-seizure recordings and 2300 seizure recordings.

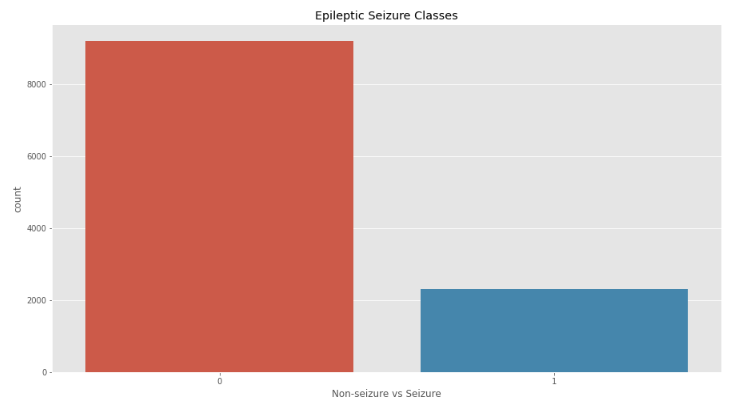


Figure 2. Epileptic Seizure Class Balance

This dataset contained no missing values, therefore, it required no imputation or removing.

*Convolution Neural Network*

Implementing a convolutional neural network (CNN) for non-image data required significant thought and consideration during the model design. As this was two-dimensional data, it required preprocessing to reshape the structure into three-dimensions. After reading the data description, which mentioned the 500 human reading was broken into 23 rows of 178, the three-dimensional shape 500 by 23 by 178 was confidently selected. By structuring the data in such a way meant that the filter sets could be parsed over a two-dimensional array, rather than a one-dimensional array. This would allow the model to learn more complex shapes in the data.

The CNN was a Keras sequential model that used a Tensorflow backend. The model consisted of two convolutional layers, two max-pooling layers, one flattening layer, and three fully-connected layers. Throughout each of the layers, a Dropout class was added to the sequence to reduce overfitting. The convolutional layers are a Conv1D class with a Rectifier Linear Unit activation function, a kernel size of 3, and a stride of 1. These parameters were used in every convolutional layer of the model. The first has a filter set of 256 while the second has a set of 128. This reduction of filters was to reduce the number of patterns the model could identify which intern would reduce model complexity and minimize overfitting. The pooling layers used a MaxPool1D class with a pooling size of 2. The three fully-connected layers used a Rectifier Linear Unit function for the activation with an exception to the output layer which used a sigmoid function to make the final binary classification. The first fully-connected layer consisted of 128 nodes leading into the second layer which contained 64.

A binary cross-entropy loss function was selected to measure model performance. It was selected for its ability to measure the distance between the prediction and ground truth for each class and then take the average for the final loss. The optimization function RMSprop was selected by hyper-tuning the model’s parameters. RMSprop is a gradient descent algorithm with momentum which prevents large steps in the vertical direction and produced large steps in the horizontal direction. This concept leads to fast convergence. This model used a batch size of 64, ran for 30 epochs, and took 3.4 seconds to execute.

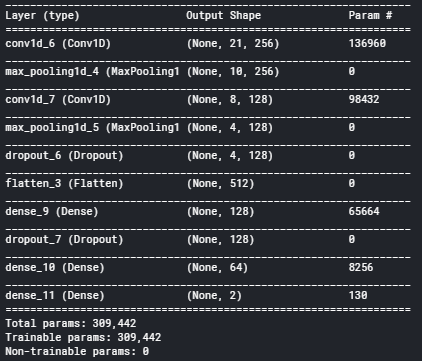


Figure 3. CNN Model Structure

The final CNN model scored a predictive accuracy of 77% against a test size of 20% of the total samples. This was a 6% reduction in training accuracy, which sat at 83%. The CNN model is the worst-performing model trained on this dataset. Its accuracy sits 14% lower than the next best performing model.

*Multilayer Perceptron*

During the design process of the Multilayer perceptrons (MLP), we were unsure of how deep to make the network as many papers offered differing opinions. To solve this problem, varying depths were trained, tested, and recorded in order to determine which architecture produced better accuracy.

Models were tested with 2, 3, 4, and 5 hidden layers. Each layer consisted of 128 nodes and contained Rectifier Linear Unit activation functions, with exception to the output layer which contained a Sigmoid function for binary classification. Each model ran for 20 epochs and had a batch size of 64. The binary cross-entropy loss function was selected for its ability to measure the distance between the prediction and ground truth for each class and then take the average for the final loss. During the testing phase, we also tested the Adam and RMSprop optimizer for each model to see which would produce the best accuracy.

With 4 architecture types, 2 optimizers, and 5 cross-validation rounds, there were 40 models to train. To reduce the manual testing load, the GridSearchCV class from the Sklearn library and the KerasClassifier class from the Wrappers library was leveraged to perform the cross-validation and optimizer tuning. This left the 4 architecture types for manual testing.

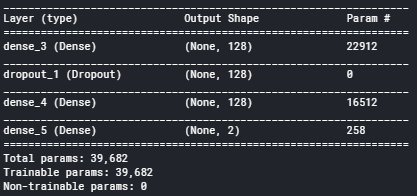


Figure 4. MLP Model Structure

The MLP model with 2 hidden layers and an Adam optimizer performed the best classification with a ROC AUC score of 0.88 while the MLP model with 5 hidden layers and an RMSprop optimizer performed the worst which a score of 0.74. This 2 hidden layer model was selected for the final MLP model. It performed a 94.7% prediction accuracy for the unseen testing data while scoring a 95.4% prediction accuracy for training data. We can be confident that this model generalizes well to unseen data. When considering the confusion matrix we see 1817

samples were correctly predicted to have not been a seizure while 356 were correctly predicted as seizures. The model predicted 18 false-positive meaning 18 samples were classified as seizures when in fact they were non-seizures. There were 109 samples classified as non-seizure readings when in fact they were. From this, we can conclude a patient EGG reading classified by this Multilayer Perceptron model has a 5% chance of being a seizure if it is labeled non-seizure.

This model was the second strong performer for this data set and executed in 13.59 seconds.

*K-Nearest Neighbors*

In order to achieve a model that has high predictive accuracy and good generalizability, it is important to adjust the model’s configuration parameters and measure how well it performs predictions. The K-Nearest Neighbors algorithm has four key parameters that can be altered to improve classification accuracy and reduce model overfitting.

The best method for testing multiple parameter combinations is using the GridSearchCV class in the Sklearn library. This class takes a base model, in this case, the KNearestNeighbors model, and a dictionary key-value pair representing the different parameters and their values. The returned object is the best performing model.

The final tuned model scored a classification accuracy of 91% for the test data set and 94% for the training set. From this, we can see this model performs well on unseen data and has good generalizability. When considering the confusion matrix we see 1835 samples were correctly predicted to have not been a seizure while 293 were correctly predicted as seizures. The model predicted 0 false-positives which means it is not classifying non-seizure samples as seizures, however, there were 172 samples classified as not being seizures when in fact they were. This is a concern as the number of false-negatives (is a seizure and classified not a seizure) is just over half the size of the true-positive classifications (is a seizure and classified a seizure). Thus, we can conclude a patient EGG reading classified by this K-Nearest Neighbors model has an 8% chance of being a seizure if it is labeled non-seizure.

This model was the second strong performer for this data set and executed in 0.13 seconds.

*Light Gradient Boosting*

Similar to the K-Nearest Neighbors model, the Light Gradient Boosting algorithm has a number of parameters that require adjusting when in search of the best model. Much like KNN, the best method for testing multiple parameter combinations is using the GridSearchCV class in the Sklearn library. This class takes a base model, in this case, the Light Gradient Boosting model, and a dictionary key-value pairs representing the different parameters and their values. The returned object is the best performing model.

When considering the confusion matrix we see 1803 samples were correctly predicted to have not been a seizure while 437 were correctly predicted as seizures. For the false-positive measurement, the model classified 16 samples as seizures when the actual class was non-seizure. There were 44 samples classified as non-seizure readings when in fact they were. This is the lowest false-negative score for all the models. From this, we can conclude a patient EGG reading classified by this Light Gradient Boosting model has a 2% chance of being a seizure if it is labeled non-seizure.

**The Broken Machine**

The Broken Machine data set contains data collected from production machines in the industry. The dataset totals 900,000 rows and 59 columns. The dataset, once loaded into memory, was 398.3MB but after memory reduction, it was reduced to 129.6MB. This is a 68% reduction in size.

The predictor variables are numeric values between 273202.5 and -480 with a mean of 5.2. The target variable is a categorical variable that consists of 2 classes:

* Class 0: machine readings indicate operating normally
* Class 1: machine readings indicate breakdown

The target class 0 contains 623,940 samples while the target class 1 contains 276060 samples.

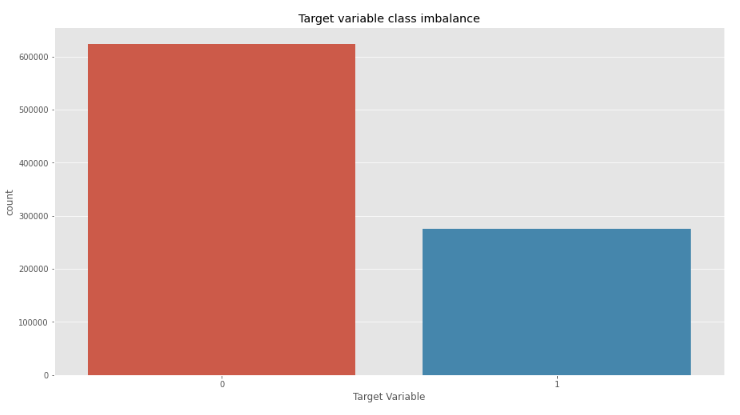


Figure 4. The Broken Machine class imbalance

*Convolution Neural Network*

The CNN model architecture used for this data set is the same CNN model architecture used in predicting the Epileptic data set. This model was tuned using the same method as the Epileptic model producing slightly different results. Kernal\_size and batch\_size remained the same at 3 and 64 respectively, however, the optimizer selected for the final model was the Adam optimizer.

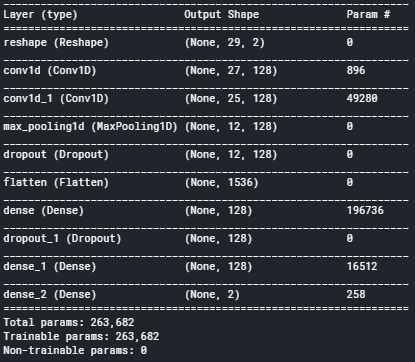


Figure 5. CNN model structure

The final CNN model scored a predictive accuracy of 69% against a test size of 20% of the total samples. The CNN model scored the same accuracy measurement as the MLP model but was 4% off the best performing model.

When considering the confusion matrix we see 187,049 samples were correctly predicted to have normal operation while 0 were correctly predicted as experiencing a breakdown. This is extremely concerning as we know there are samples that have broken down and for the model to predict 0 breakdowns means it is not a model that should ever be used in production. For the false-positive measurement, the model classified 0 samples as breakdowns when the actual class non-breakdowns. There were 82,951 samples classified as non-breakdown readings when in fact they were.

This model was the second strong performer for this data set and executed in 488 seconds.

*Multilayer Perceptron*

The architectural structure of the MLP model is identical to that of the Epileptic MLP model. It consists of two fully-connected Rectifier Linear Unit layers that contain 128 nodes and one fully-connected Sigmoid layer that produces the predictions. The tuning process selected slightly different parameters. Batch\_size remained the same at 64, however, the Adam optimizer and 50 epoch rounds were selected for the final model.

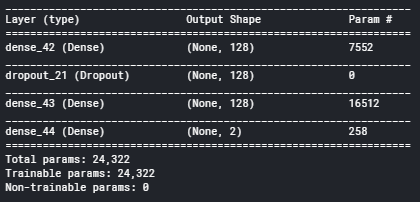


Figure 5. MLP model structure

The final MLP model scored a predictive accuracy of 69% against a test size of 20% of the total samples. The MLP model scored the same accuracy measurement as the CNN model but was 4% off the best performing model.

When considering the confusion matrix we again see 187,049 samples were correctly predicted to have normal operation while 0 were correctly predicted as experiencing a breakdown. After seeing two results the same for two different models, we would have to consider that there is an error with the configuration of our CNN and MLP models. For the false-positive measurement, the model classified 0 samples as breakdowns when the actual class non-breakdowns. There were 82,951 samples classified as non-breakdown readings when in fact they were. This model is useless and should not be used to predict machine breakdowns.

This model was the second strong performer for this data set and executed in 180 seconds.

*K-Nearest Neighbors*

The KNN model scored a classification accuracy of 65.8% for the test data set and 71% for the training set. These differences show a good model generalizability, however, this accuracy is not suitable for production classification. When considering the confusion matrix we see 113,112 samples were correctly predicted to have not been as normal machine operation while 5163 were correctly predicted as break-downs. The model predicted 50030 false-positives as breakdowns when the actual class was non-breakdowns.

There were 11695 samples classified as not being non-breakdowns when in fact they were. This means the K-Nearest Neighbors model has a 30% chance of being a breakdown if it is labeled normal operation.

This model was the second strong performer for this data set and executed in 114 seconds.

*Light Gradient Boosting*

When considering the confusion matrix for the LGB model we see 120,058 samples were correctly classified as normal machine operation while 12,596 were correctly classified as breakdowns. For the false-positive measurement, the model classified 4763 samples as breakdowns, when the actual class was non-breakdowns. There were 42583 samples classified as normal machine operation when in fact they were breakdowns. This is the lowest false-negative score for all the models. From this, we can conclude a machine’s operational data when classified by this Light Gradient Boosting model has a 26% chance of being a breakdown if it is labeled as operating normally. This is an accuracy that would deem this model unreliable for production implementation.

This model was the second strong performer for this data set and executed in 16 seconds.

**Dataset: SVHN**

SVHN is a real -word image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MINST but incorporates an order of magnitude more labeled data (over 600,000-digit images) and comes from a significantly harder, unsolved, real world problem i.e. recognizing digits and numbers in natural scene images. SVHN is obtained from house numbers in Google Street View Images.

It consists of 10 classes, 1for each digit. Digit ‘1’ has label1, ‘9’ has label 9 and ‘0’ has label 10.

It is found that the data are divided into two formats and in this assignment, we used a second format.

Format 1: Original images with character level bounding boxes.

Format 2: MINST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides)

(73257,32,32,3) digits for training, (26032,32,32,3) digits for testing.

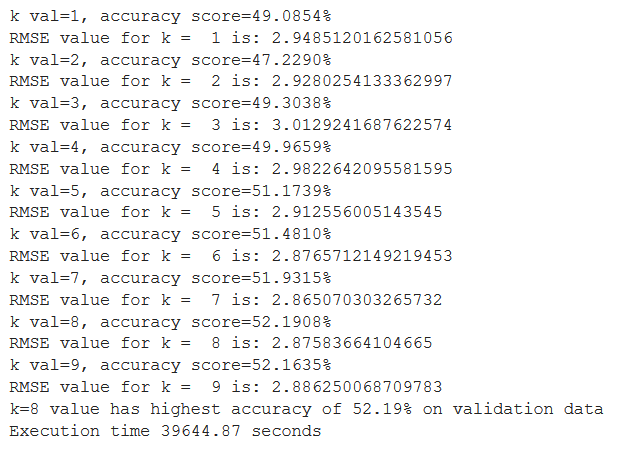
For model designing purposes we used MINST-like 32-by-32 images centered around a single character.

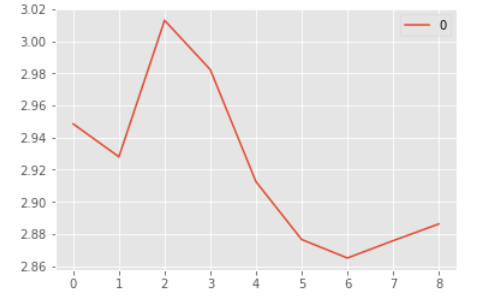
*Shallow Learning Models: k-NN (k-Nearest Neighbors):*

For the K-NN algorithm here we convert RGB images into Gray images. The shape of train data is (x=73257,1024) and (y=73257,1) and for test data is (x=26032,1024) and (y=26032,1) .The validation data is 20% of the train set.

The k-NN model has been tested using 1 to 10 neighbors (k value) and found that the Root Mean Squared Error (RMSE) is low using 8 neighbours. The final model was trained using 6 neighbours and uniform weights.

Following is the accuracy score for different values of k from this it is clearly shown that k=8 shows good accuracy. Plot between k values and RMSE.

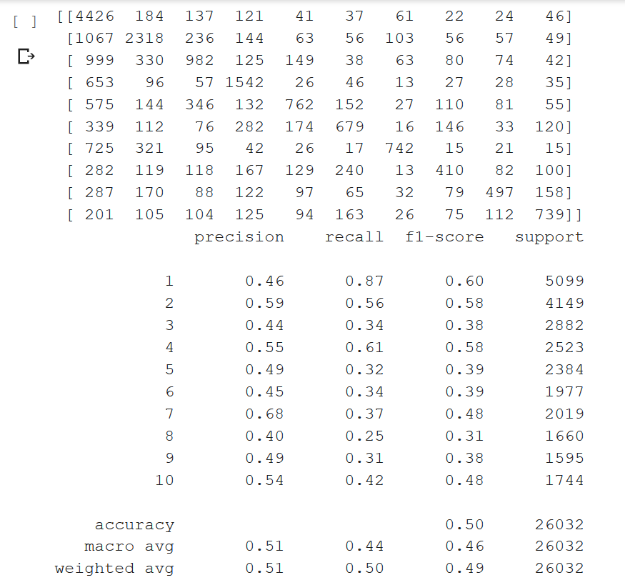
****

****

Test Acc = 0.5031

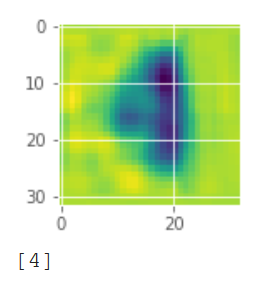
Val Acc = 0.5219

Train Acc = 0.6451



The above figure shows confusion matrix and classification report from which one can find that prediction of 1,2 and 4 has more score (can detect more accurately than other digits)

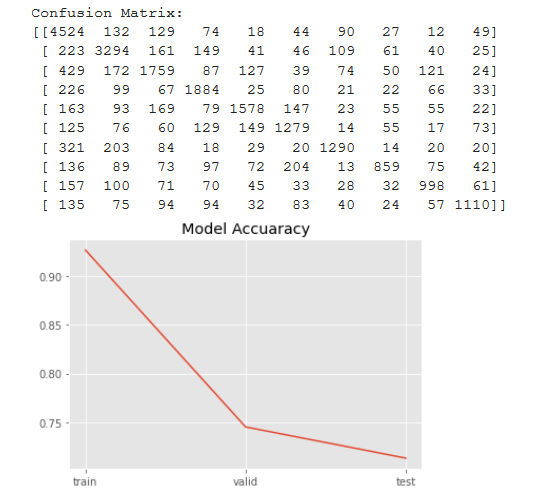
**Visualization**: As per f1\_score model predict 4 more accurately.



*LGBM*

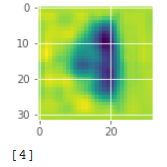
To pass the data to a shallow learning model, this RGB image data with the size 32\*32\*3 has converted into a gray image. The LGBM classifier ran using 10 early stopping rounds by evaluating train and validation data for a maximum of 200 estimators. Time taken to run the model is 946.16 Sec.

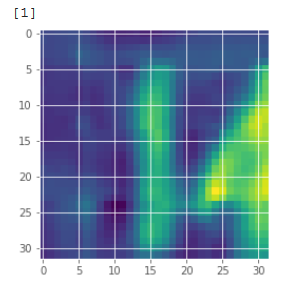
The training accuracy of the model is 0.9264, the validation accuracy is .7454 and the test accuracy is .7135.



It has been found that from the confusion matrix the number of images of class 1, 2 and 4 is more than the rest of class. Hence while predicting classes other than 1,2, and 4 it gives incorrect results.

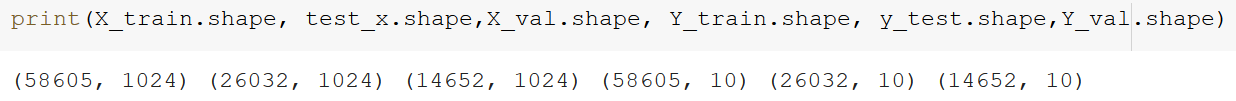
**Visualization:**

****

****

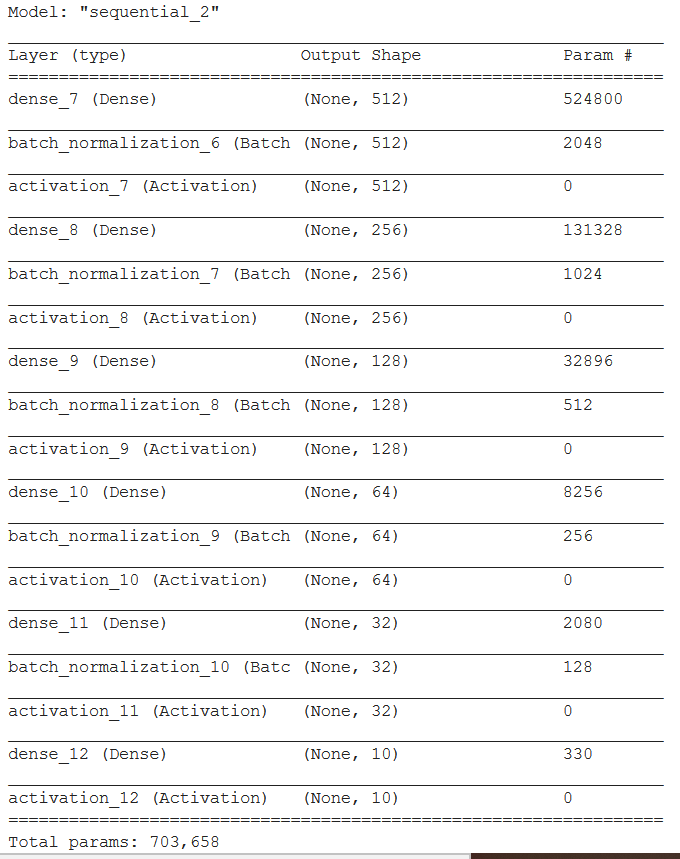
*MLP (MultiLayer Perceptron):*

In our first deep learning model we used a Gray image with the following shape of data.

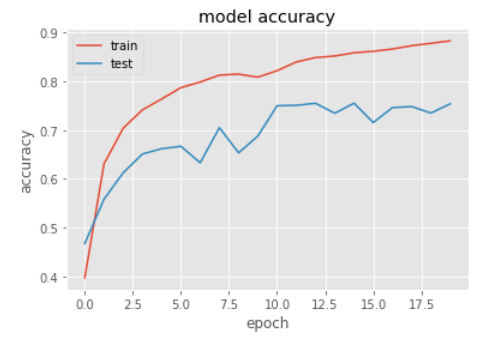


This is an example of supervised learning, and is carried out through backpropagation. As the SVHN dataset is an example of multiclass classification we used the following functions for getting reliable results. This model has been designed with three layers (an input, an output layer with 4 hidden layers) with *relu* and *softmax* activation functions which outputs a vector that represents the probability distribution of a list of potential outcomes. It has been compiled with categorical cross entropy and sgd optimizer.

Model Summary:

****

**Accuracy**

****

train acc= 0.7931

test acc= 0.7203

val acc= 0.7552

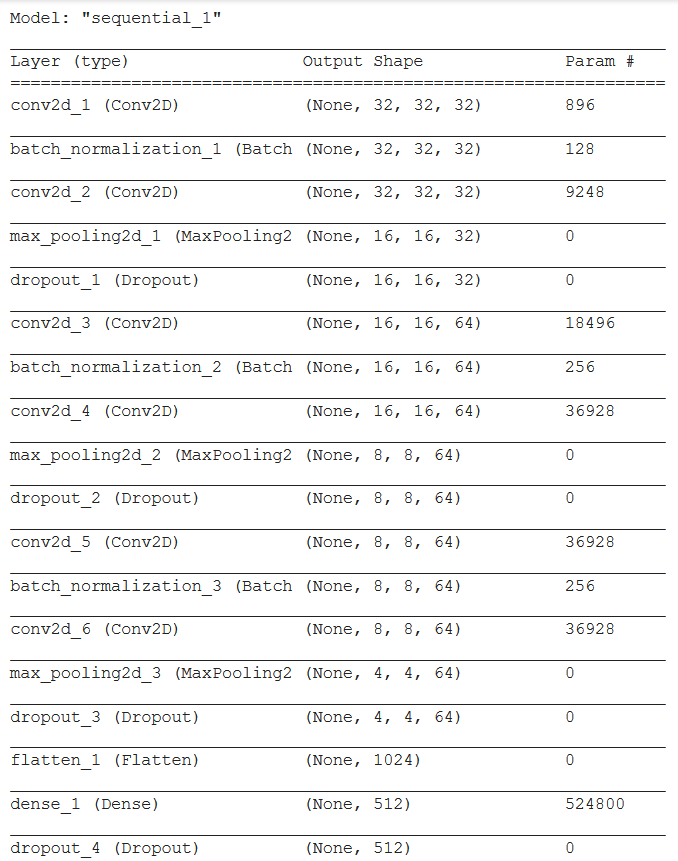
**Visualization:**



*CNN (Convolutional Neural Networks):*

For CNN we used here format two i.e. RGB images dataset. The first step is to fix axes of image i.e. earlier the shape of image is (32,32,3,73257) which turned into (73257,32,32,3) for both train and test data. Normalizes the data and then one hot encoding of labels is done as the labels contain class. Here also we consider validation data as 20% of train data.

Model Summary:



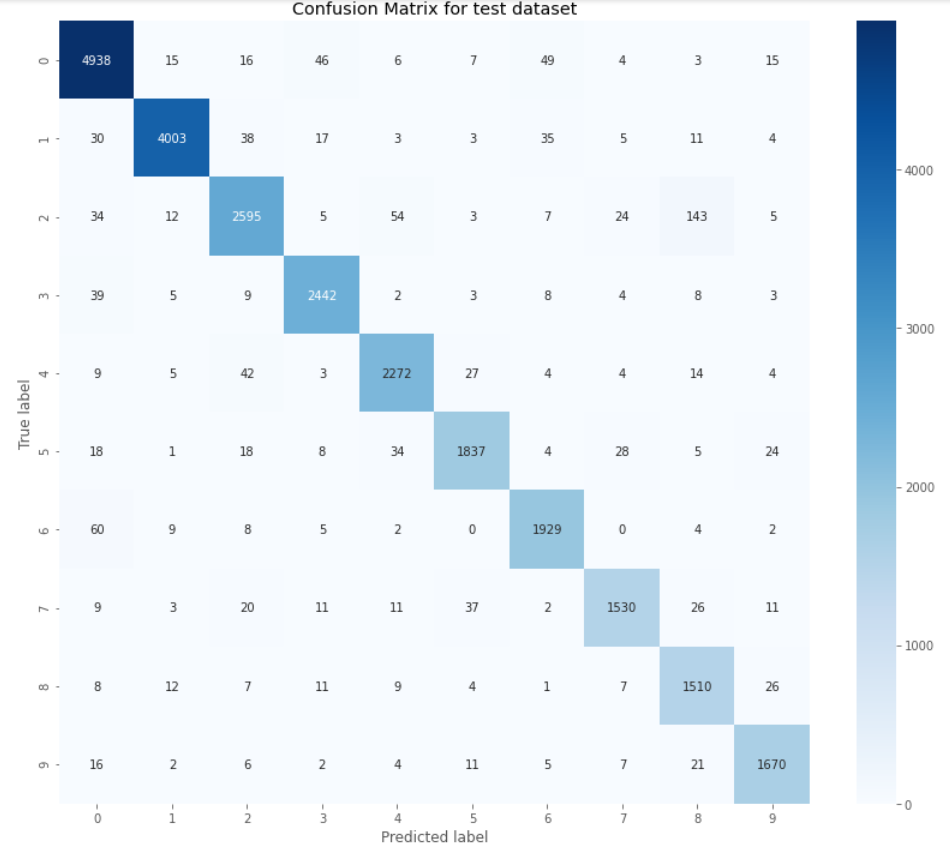
To get a more accurate result here we augmented the images in the data set, by rotating, zooming in and out, shifting them up and down and shearing them. For a good learning rate of model optimizer here we use the AMSGrad variant. Selected learning rate lr=0.01 for stable loss. We set a callback in an auxiliary model which will gradually increase the learning rate of the optimizer. Epoch set to 50 and batch size equal to 100.

Test Accuracy = 0.9498

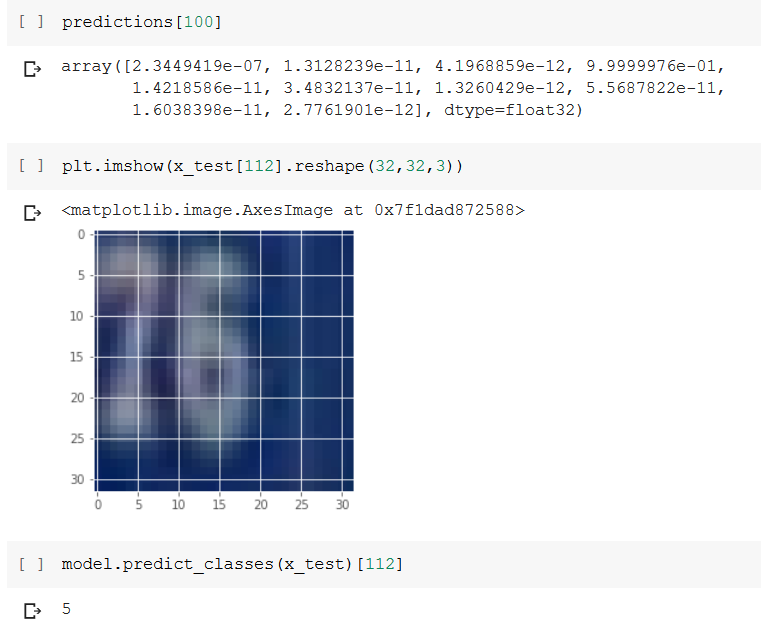
Train Accuracy = 0.9724

Val Accuracy=0.9462

Confusion Matrix :



**Visualization:**

****

*Autoencoder for Image Reconstruction*

With the help of Neural Network an autoencoder can learn how to decompose data (Image) into small bits of data, and then using that representation, reconstruct the original data as closely as original.

Encoder: Learns how to compress the original input into a small encoding.

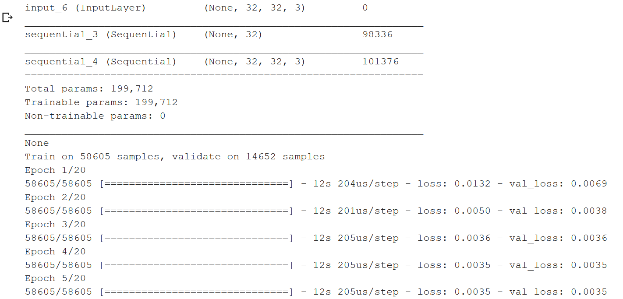
Decoder: Learns how to restore the original data from that encoding generated by the encoder.

Our data is in the form of a 3D matrix (x\_train, RGB images). These images will have large values ranging from 0 to 255. It is needed to normalize images to train models faster and get better results.

We split the data into tests by giving a ratio and the remaining one is training data. Random state is assigned with a certain value so that it produces the same result, even though running code several times.

The autoencoder sequential takes two parameters i.e. image\_size (32, 32, 3) and code\_size. The smaller the code\_size, the more the image will compress, but less features will be saved and the reproduced image will be that much more different from the original. The InputLayer has an input vector – image\_size. The flatten layer’s work is to flatten matrix into a 1D array. The last layer Dense layer is used to find the optimal parameters that achieve the best output i.e. encoding and set output size to the code\_size.

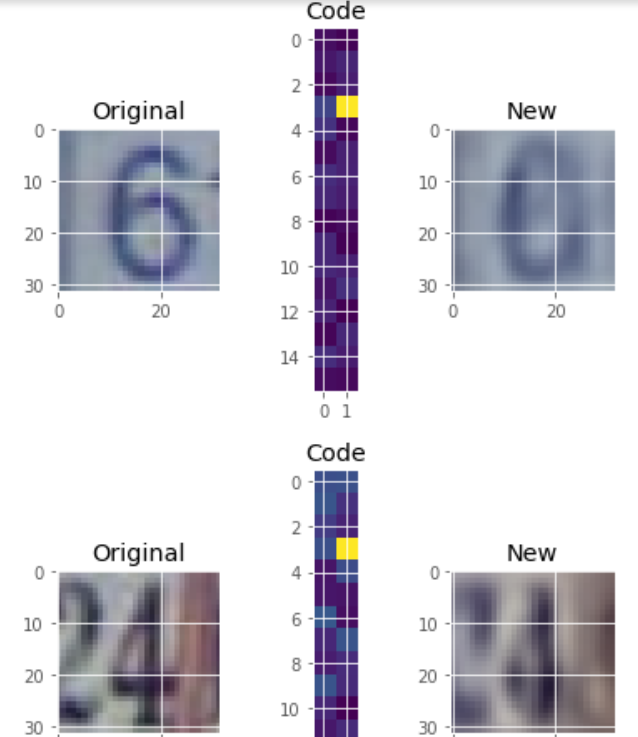
The epochs variable defines how many times the training data to be passed through the model and the validation\_data is the set use to evaluate the model after training.



The decoder is also a sequential model and it accepts the input and tries to reconstruct it in the form of a row. Later on, it stacks it into a 32x32x3 matrix through the Dense layer. The final Reshape layer will reshape it into an image.



**Visualization**



**Dataset: Forest Cover**

This dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. All observations are cartographic variables from 30-meter x 30-meter sections of forest. There are over half a million measurements total. The dataset includes information on tree type, shadow coverage, distance to nearby landmarks (roads etc), soil type and topography. With the help of k-NN we tried to find out solutions to the following questions.

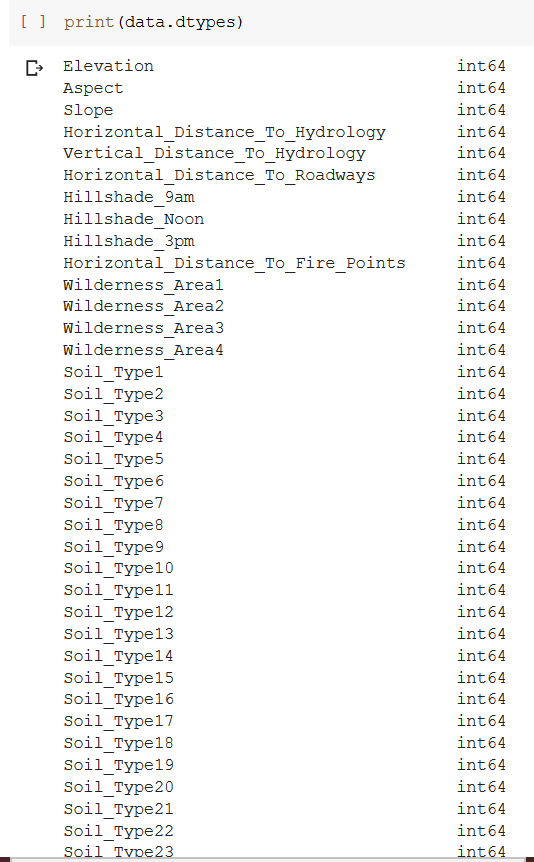
1.Can you build a model that predicts what types of trees grow in an area based on the surrounding characteristics?

2.What kinds of trees are most common in the Roosevelt National Forest?

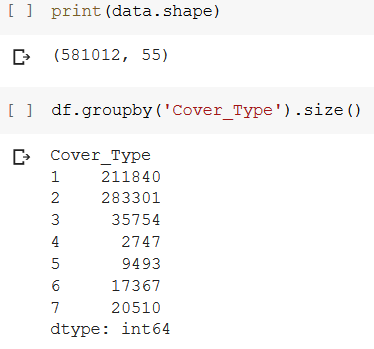
3.Which tree types can grow in more diverse environments? Are there certain tree types that are sensitive to an environmental factor, such as elevation or soil type?

We have studied some more shallow models for comparison purposes.

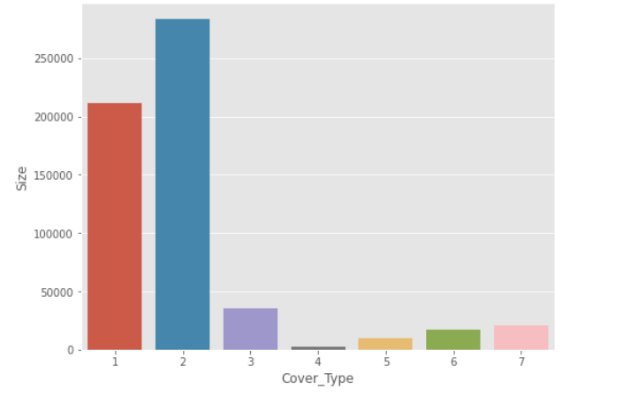
Following are Data types of the dataset.



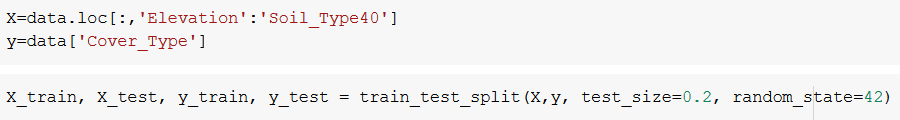
There are seven types of cover in which a number of trees are mentioned. There are 55 columns in which mostly are soil type columns.



Following is the graph between cover and size of trees.

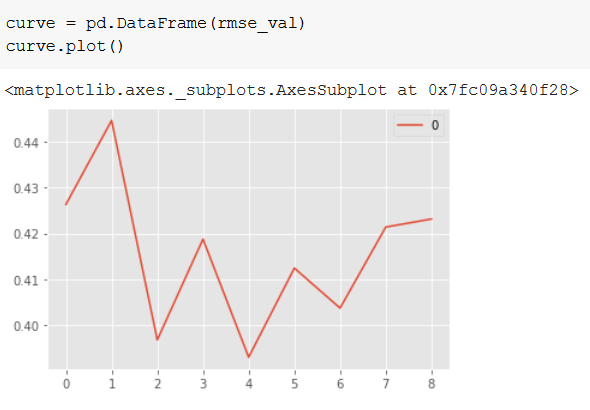


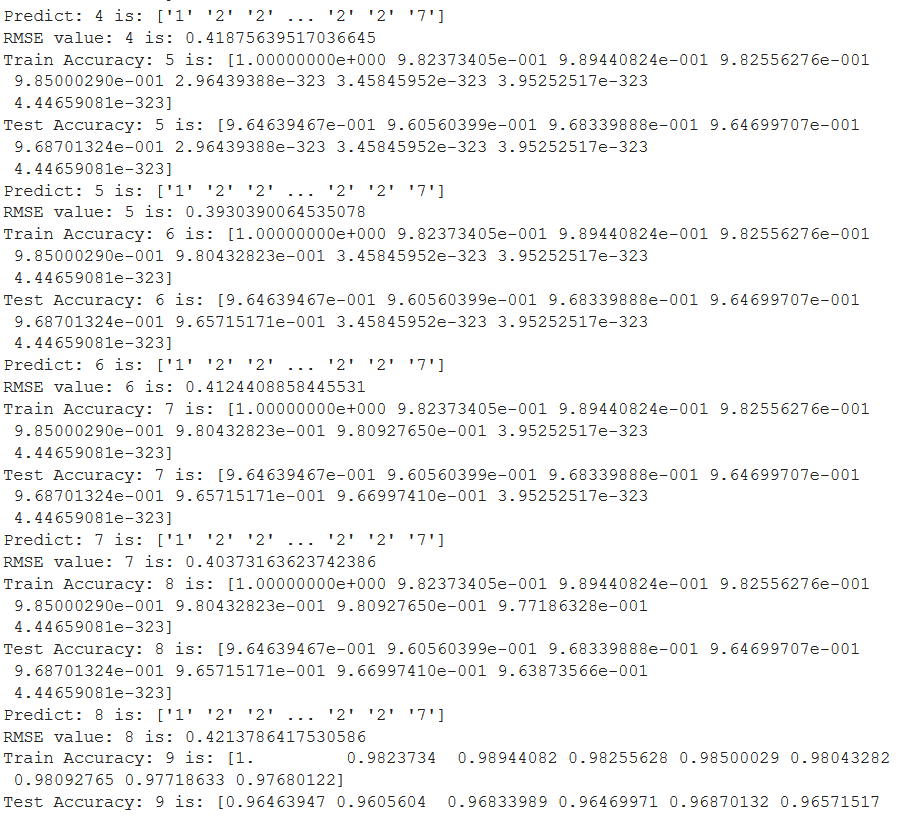
Following is the distribution of the dataset into train data and test data. Here from column Elevation to soil type we are predicting cover\_Type.

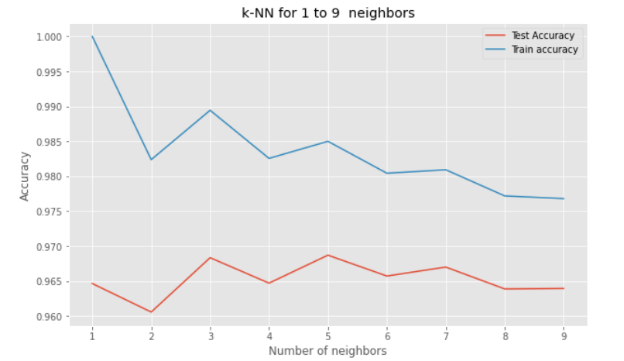


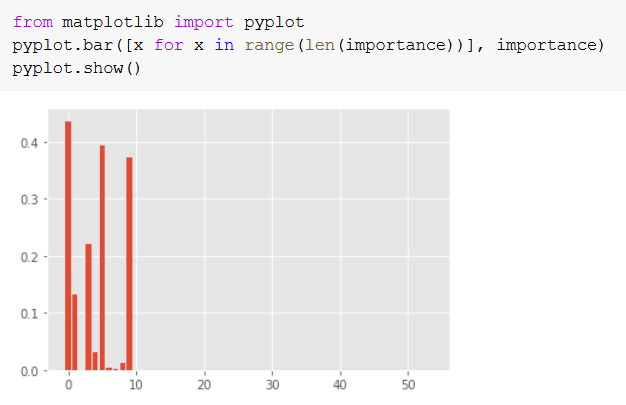
*K-Nearest Neighbors*

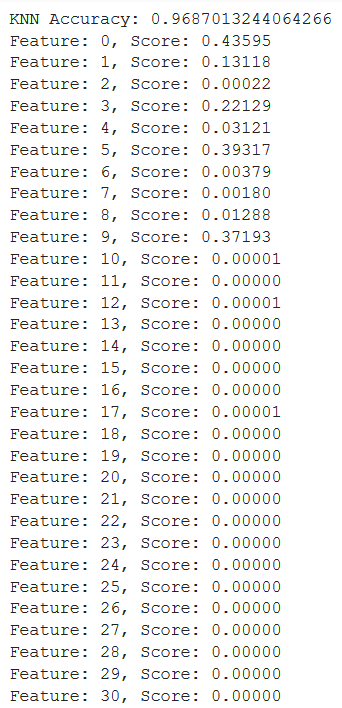
To get accurate types of tree growth in an area based on the surrounding characteristics we tried different k values ranging from 1 to 10. From this experimental set up we got to know k = 5 gives best accuracy comparatively to other values. Following are results of different values of k and plot of RMSE for different values of k.











From the above result we get to know which feature is responsible for the type of trees.

**Following are the comparative results with different models**.

|  |  |
| --- | --- |
| Model | Accuracy |
| GaussianNB | 45.6881 |
| RandomForest Classifier | 56.8092 |
| Linear SVC | 60.4434 |
| Logistic Regression | 62.2299 |
| Gradient Boosting Classifier | 77.2872 |
| DecisionTree Classifier | 93.927 |
| k-NN Classifier | 96.8701 |

*Light Gradient Boosting Model (LGBM):*

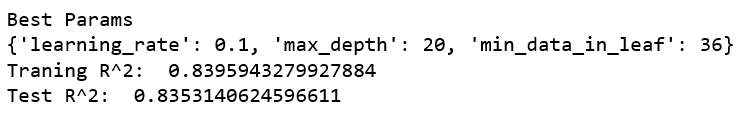
While using data for different models the first step we used to reduce memory usage to 13.8889% percent of initial memory usage, earlier it was 239.3696 MB to 33.24589 MB. The hyper parameters of the LightGBM have been tuned by giving different values to learning rate, max depth and to the min data in leaf. The learning rate controls how quickly the model is adapted to the problem. In general a low learning rate with high number of iterations would improve the performance of the model. *Max Depth* is another hyper parameter which specifies the max depth to which each tree will be built. In general, deeper trees will be prone to overfit because deeper trees learn more about the training data so couldn’t generalise well enough. Another interesting hyper parameter is ‘*min data in leaf’*, this is another parameter to control the depth of the tree. This represents the minimum observations in a leaf node to split further. Initial given parameters to model are:

learning \_rate : np.logspace(-3, 0, 4)

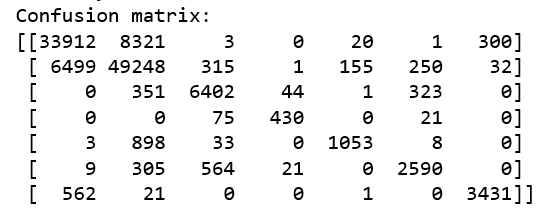
max\_depth : np.linspace(10, 40, 4, dtype=np.int)

min\_data\_in\_leaf : np.linspace(20, 100, 6, dtype=np.int)

Model has been attempted with all the combinations of parameters using grid search cv to find the best params for this dataset which has been evaluated by the accuracy . Following are the best parameters with accuracy and execution time.





ConfusionMatrix: 

Here for class 1, out of 42,557 data points 33,912 were correctly classified. and 8321 were classified as class 1. Intended for class 1, out of 56,500 samples, 49248 were correctly classified. and so on.

*MLP (MultiLayer Perceptron):*

This model has been designed with two fully connected dense layers with *relu* and *softmax* activation functions and also uses dropouts to avoid overfitting and reduce the parameters. Model also tried using different numbers of layers and hidden units. It has been compiled with categorical cross entropy and adam / rmsprop optimizer for experimental purpose, with init\_mode = [uniform, glorot\_normal, glorot\_uniform] and epoch = 30 , 50 to find best accuracy for forest cover data.

Model has been attempted with all combinations of parameters using grid search cv to find the best params for this dataset which has been evaluated by the accuracy.

The following are the best parameters for the MLP model for a given dataset.



The accuracy found and execution time for forest cover data using MLP model.





*CNN (Convolutional Neural Networks):*

A Sequential model has been built using various layers. The data has been scaled between 0 and 1 and passed to the Keras convolutional 1d layer. This model used relu and sigmoid as an activation function. Model has been tuned using different layers, hidden units and activation functions.

Model ran using the oversampled data with the batch size of 64 and used early stopping rounds of 50 epochs.

The best parameters are as follow

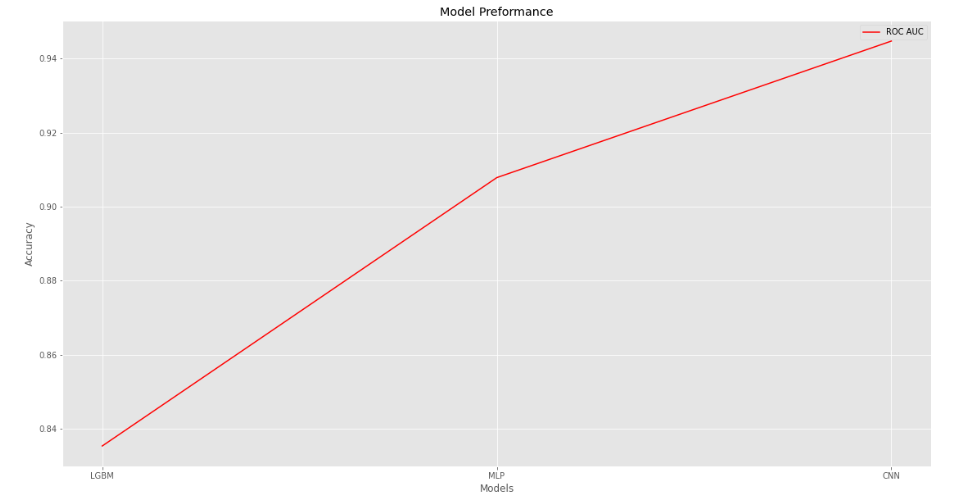


Accuracy of model and execution time:





Following graph shows model performance on three different algorithms.



**German Traffic Signs:**

### Context

Detecting Street Signs is one of the most important tasks in Self Driving Cars. This dataset is a benchmark in Street signs and symbols including 43 different classes. Classifying road symbols is the aim of this dataset.

**Data**: This data has three separate sets: train, validation and test sets.



This is an image dataset with 32\*32 size and has three channels such as red, green and blue. The target is to predict 43 different classes so it's a multi-class classifier.

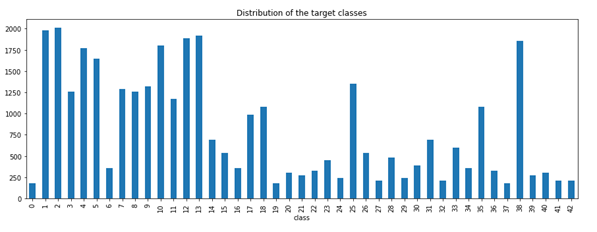
**Visualisation**:

Displaying a few different class images with the labels to sense the data.



The picture quality is quite good for some images, some images are very dark and hard to recognise. We can see some images are zoomed out and some are zoomed in. These images have been taken while the car is moving, so can expect different ranges and the clarity of the images could be varied as they could have been taken in the daytime, night-time, sunny day, foggy day or a rainy day.

**Target Distribution:**

****

The classes are imbalanced in this dataset, some signs have up to 2000 images to learn from and some have only less than 250; those might be harder to learn from.

**Scaling:**

The image pixel values are in the range between 0 to 255. To improve the training time of the model, we are scaling the pixel values to the range 0 and 1.

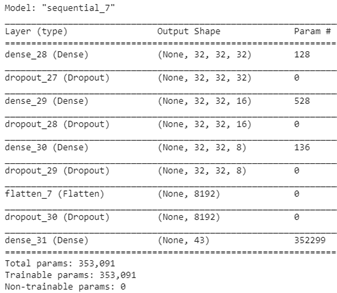
**Deep Learning Models:**

**MultiLayer Perceptrons (MLP):**

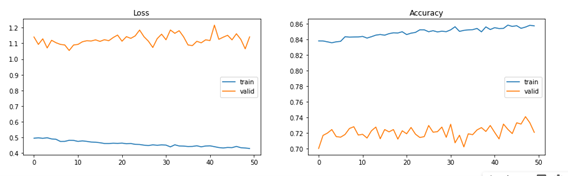
The model has been tried with different numbers of hidden layers and nodes to reduce overfitting and improve the accuracy. The final model has 4 dense layers, three of those used *relu* activation function and *softmax* for the last one. Dropout layer has been used after each dense layer to reduce overfitting. Flatten layer has been used to convert the 3-dimensional data into a vector. *sparse\_categorical\_crossentropy* has been used as a loss function because it’s a multiclass classification task and targets are integers.

This model ran with *50 epochs*, validation dataset accuracy has been used as a monitor to the early stopping rounds.

Here is the model summary:



Below is the plot which shows the loss and the accuracy of the model:



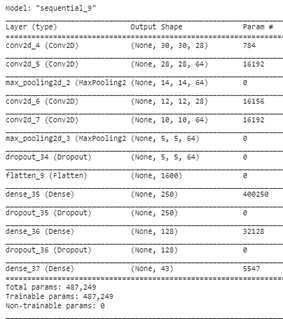
Validation accuracy is poor and loss is high compared to the training accuracy. This model is *overfitting* the data.

**Convolutional Neural Network (CNN):**

A medium sized model has been used and tuned which has given .96 accuracy. These images are taken from different ranges and angles so the augmentation might have been helpful. Augmentation using keras imagedatagenerator has been used to rotate, width shift, height shift and zoom range to improve the model accuracy. This method decreased the accuracy of the model. When we have enough data to train, augmentation might not be very helpful that could be the reason for it.

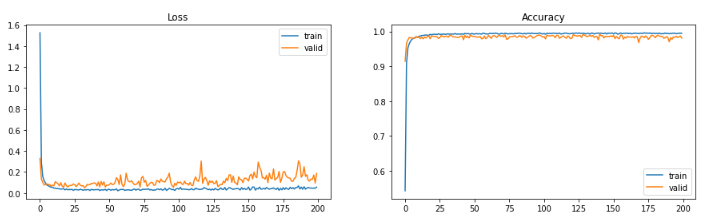
A bit larger model has been initiated and this model is used *convolutional 2d, max pooling 2d, dropout, flatten*, and fully connected *dense* layers with the activation functions *relu* and *softmax*. The Conv2D layer has been used with the kernel size of 3\*3 and *relu* activation function. The model has been tuned with a different combination of layers, hidden units and different activation functions to get better accuracy and reduce overfitting. This used sparse categorical cross entropy as a loss function as this is a multi-class classifier and Adam optimizer has been used.

This is the summary of the selected model:



Model ran with 200 epochs and using early stopping rounds as a call back with the validation set

These plots show the Loss and Accuracy of the model on the training process:



This model has given the best test accuracy of 97.7% for this dataset.

**Shallow Learning Models:**

**LightGBM:**

To pass the data to a shallow learning model, this image data with the size 32\*32\*3 has converted into a vector that has 3,072 features so the data became sparse. The LGBM classifier ran using 10 early stopping rounds by evaluating train and validation data for a maximum of 200 estimators. The model found the best iteration at round 73 that has the training multi log loss of .01825 and validation loss of 1.2879. Time taken to run the model is 3167.13 Sec.

The training accuracy of the model is 1.0, the validation accuracy is .71 and the test accuracy is .73. We can easily say this model is overfitting the data. It’s possible to tune the model using hyper parameters to avoid overfitting but it’s taking a lot of time to run different combinations of parameter settings because of having huge data with a lot of features.

**KNN (K-Nearest Neighbours):**

In the K-Nearest Neighbours Classifier, K value is highly dependent on the data set, so the classifier has been tried with a few different k values and chosen as the one with low error rate. The 5 neighbours and uniform weights have been used to generate the model.

The training accuracy of the model is .94, validation accuracy is only .324 and accuracy on the unseen test data is .38. This model is overfitting the data which means the model fits the noise in the data.

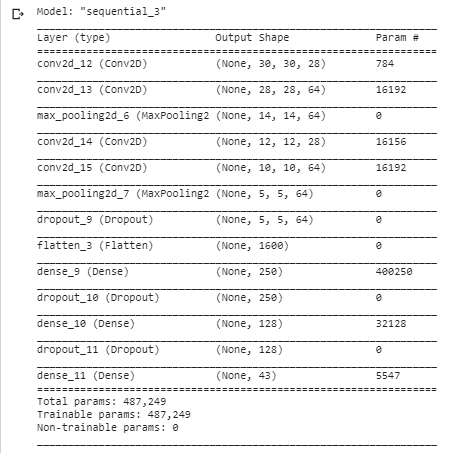
In general, the higher k value suppresses the effect of the noise, so k value has been increased and used k=15. The training accuracy of the model is .85, validation accuracy is .37 and test accuracy is .31. Even though k value increased it didn’t improve the accuracy of the test set.

Overall the model is overfitting to the data which means it fits to the noise in the data.

**Combined Model (CNN + KNN):**

The tuned CNN model has been used, which has two sets of convolutional 2D layers followed by a max pool layer to reduce the input parameters and avoid overfitting. Followed by a dropout and flatten layers then used two dense layers with a dropout layer. 3\*3 filter size has been used for convolutional layers and relu activation function has been used for the convolutional and dense layers.

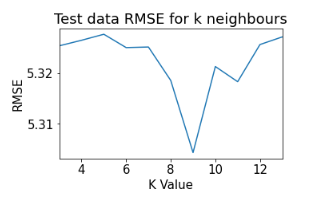
Model Summary:



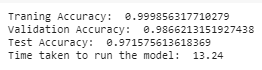
Model ran with 200 epochs and used early stopping rounds as a call back with the validation set. The model is evaluated using the batch size of 64.

Outputted the intermediate layer of the CNN model which is the dense layer(12th layer) and is used as an input to the KNN model.

The K-Neighbours classifier has been tested using a range of k values from 3 to 15. A subset data has been used to speed up the process.



K=9 has been used for the final KNN Classifier. The final accuracy of this combined model is:



This combined CNN + KNN model has given good accuracy but still a deep learning CNN model is best for this dataset.

**PolishCompanies Bankruptcy**

**Data Set Information:**

The dataset is about bankruptcy predictions of Polish companies. The data was collected from Emerging Markets Information Service (EMIS, [Web Link]), which is a database containing information on emerging markets around the world. The bankrupt companies were analysed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.

Basing on the collected data five classification cases were distinguished, that depends on the forecasting period:

- 1stYear => the data contains financial rates from the 1st year of the forecasting period and corresponding class label that indicates bankruptcy status after 5 years. The data contains 7027 instances (financial statements), 271 represents bankrupted companies, 6756 firms that did not bankrupt in the forecasting period.

- 2ndYear => the data contains financial rates from the 2nd year of the forecasting period and corresponding class label that indicates bankruptcy status after 4 years. The data contains 10173 instances (financial statements), 400 represents bankrupted companies, 9773 firms that did not bankrupt in the forecasting period.

- 3rdYear => the data contains financial rates from the 3rd year of the forecasting period and corresponding class label that indicates bankruptcy status after 3 years. The data contains 10503 instances (financial statements), 495 represents bankrupted companies, 10008 firms that did not go bankrupt in the forecasting period.

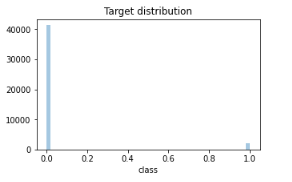
- 4thYear => the data contains financial rates from the 4th year of the forecasting period and corresponding class label that indicates bankruptcy status after 2 years. The data contains 9792 instances (financial statements), 515 represents bankrupted companies, 9277 firms that did not bankrupt in the forecasting period.

- 5thYear => the data contains financial rates from the 5th year of the forecasting period and corresponding class label that indicates bankruptcy status after 1 year. The data contains 5910 instances (financial statements), 410 represents bankrupted companies, 5500 firms that did not bankrupt in the forecasting period.

**EDA**

This dataset has 5 years of data. These separate datasets have been combined and created a column called ‘year’ to distinguish the data. The shape of the data is (43405, 66) so it has 65 predicted variables and one target variable. The target variable ‘class’ has been converted to an unsigned integer. There are a couple of missing values which have been replaced with zero. The data plotting for different variables shows that some features have -ve values, some variables have mostly zero values.

Target distribution:

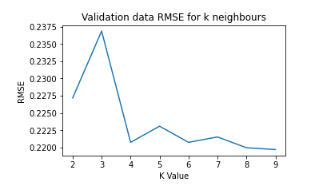


Standard scaler has been used to transform the data to unit variance. Data was split into train, validation and test sets with respectively 60%, 20%, 20%.

**Shallow Learning Models:**

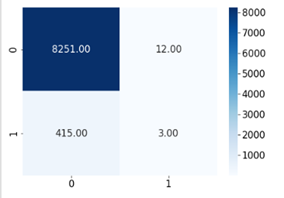
**KNN (K-Nearest Neighbours):**

The KNN model has been tested using 2, 10 neighbours (k value) and found that the Root Mean Squared Error (RMSE) is low using 6 neighbours. The final model was trained using 6 neighbours and uniform weights.



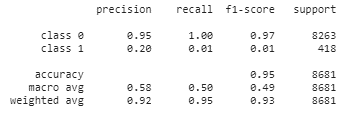
The train, validation and test accuracy of this model is .95. This is an unbalanced dataset, so the model is predicting towards one class. The model estimation is that most of the company’s bankruptcy status is 0 which means only very few companies had bankrupted.

Plotted confusion matrix:



For class 0, out of 8,263 samples, 12 were incorrectly classified and 8,251 were correctly classified. For class 1, out of 418 samples, only 3 samples were correctly classified as class 1 and 415 were incorrectly classified.

Classification Report:



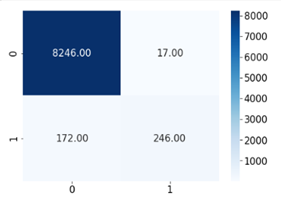
The recall and f-scores are .01 for class 1. Recall is the fraction of the total amount of relevant values that the model correctly identified/classified. This model only be able correctly classify 1% of class 1 values that means the model will only be able to identify 1% of bankrupt companies.

Even though the accuracy of the model is good, it’s a useless model for this dataset.

**Light Gradient Boosting Model (LGBM):**

The hyper parameters of the LightGBM have been tuned. The bagging is bootstrap aggregation which is generally used in the decision trees to reduce overfitting and improve the accuracy of the model. *Bagging fraction* .7, .8 and .9 has been used to find the best fraction for this dataset and used .9 in the final model. The *learning rate* is another hyper parameter which is in range between 0 and 1. The learning rate controls how quickly the model is adapted to the problem. In general, a low learning rate with high number of iterations would improve the performance of the model. This model has been tested with the learning rates of [.02, .2, 1] and chosen .2 as a suitable one for this problem. *Max Depth* is another hyper parameter which specifies the max depth to which each tree will be built. In general, deeper trees will be prone to overfit because deeper trees learn more about the training data so couldn’t generalize well enough. To determine an appropriate value, the model has been validated by using the max depths of 10, 20, 25 which resulted in 20 is appropriate one for this dataset. Another interesting hyper parameter is ‘*min data in leaf’*, this is another parameter to control the depth of the tree. This represents the minimum observations in a leaf node to split further. This parameter has been tried with [20, 30, 40, 60, 80] and resulted in 80. One more hyper parameter is *reg alpha* that is a regularization term on weights, used [1, .8, .5, 0] to find the optimal value and get .8 as the best. Model has been attempted with all the combinations of parameters using grid search cv to find the best params for this dataset which has been evaluated by the validation accuracy and early stopping rounds has been used to find the best iteration.

Cross validation Matrix:



Here for class 1, out of 8,263 data points 8,246 were correctly classified and only 17 were classified as class 1. Intended for class 1, out of 418 samples, 246 were correctly classified.

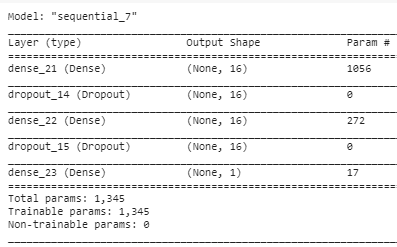
LGBM has given best accuracy for this dataset so k-fold has been implemented to get better accuracy. K-Fold has been implemented with 11 folds and taken the best iteration at each round to predict the test data. Each fold ran up to 1000 estimates and used early stopping of 100 rounds. This classifier also used *scale position weight* of 20 to balance the unbalanced classes. From the 11 folds the majority class has been considered for each instance. This method has given the best test accuracy of .9816

**Deep Learning Models:**

**MLP (MultiLayer Perceptron):**

This model has been designed with three fully connected dense layers with *relu* and *sigmoid* activation functions and also uses dropouts to avoid overfitting and reduce the parameters. Model also tried using different numbers of layers and hidden units. It has been compiled with binary cross entropy and adam optimizer.

Model Summary:

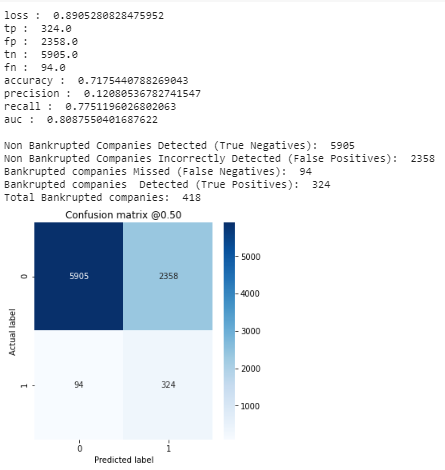


As this is an unbalanced dataset, initial bias has been created using log(num of positive samples/num of negative samples) calculation, this calculated bias has been passed to the model and saved the initial weights from this model. These careful initial weights are used to fit the model but still the model didn’t manage to predict the minor class instances.

Weights are calculated and passed by using keras class\_weight parameter. Even the weighted model didn’t manage to identify bankrupted companies at all.

The SMOTE oversampling method has been used to create a balanced dataset. Oversampling is a method to increase the minor class samples to match with the major class. The model with the balanced data set has fitted using early stopping rounds that validates using validation set and can run up to 100 epochs. This model managed to identify some bankrupted companies effectively.

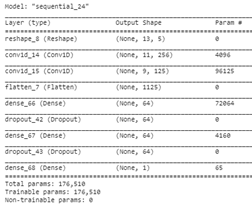
Confusion Matrix and evaluation details:



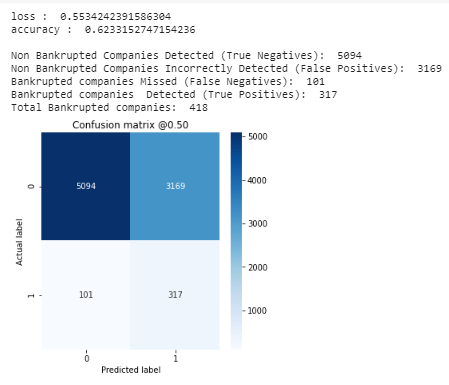
**CNN (Convolutional Neural Networks):**

A Sequential model has been built using various layers. The data has been reshaped by adding a reshape layer as a first layer to the model to get an accepted shape of the data to pass to the Keras convolutional 1d layer. This model used sigmoid as an activation function. Model has been tuned using different layers, hidden units and activation functions.

Here is the model summary:



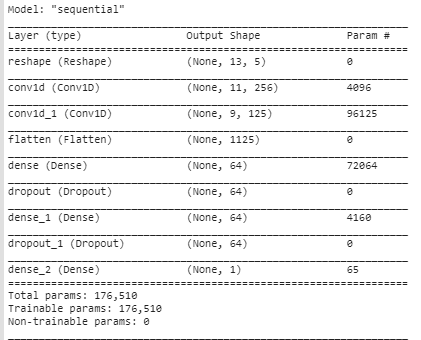
Model ran using the oversampled data with the batch size of 2048 and used early stopping rounds of 100 epochs. This model managed to identify the bankrupt companies very well, but a false positive rate is high that means it flagged lots of non bankrupt companies as bankrupt.

.

**Combined Model (CNN + KNN):**

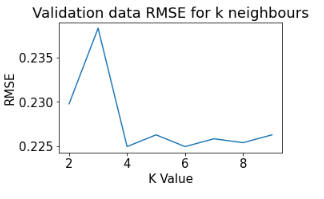
A sequential CNN model has been used, the initial layer of the CNN is a reshape layer to convert the input shape into two dimensions. Two conventional 1d layers with the filter size of 3 and a sigmoid activation has been used. A flatten layer and three dense layers and each dense layer with a dropout layer. Binary cross entropy as a loss function and Adam optimizer has been used.

Model Summary:

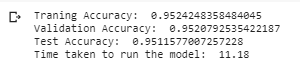


Intermediate layer of this CNN model which is the dense layer (5th layer) output has been taken out.

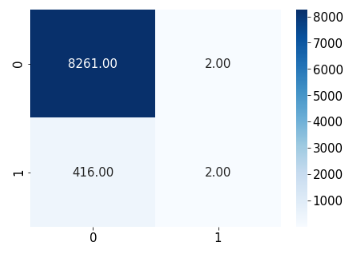
This CNN model intermediate layer output parameters have been used as an input to the KNN model. The KNN model had been tried with the different k values.



K= 6 has been used for the final KNN Classifier. Here are accuracies of this combined model:



Heatmap of the Confusion Matrix:



True positive rate is only 2 which means this model is only able to identify 2 bankrupted companies correctly and 416 of those were incorrectly classified as non bankrupted.

This model didn’t manage to identify the bankrupted companies so this is not a good model for this dataset.

Overall the Light Gradient Boosting Model has given the most accurate predictions for this dataset.

**Conclusion**

A visual examination of table 1 reveals how well each model scored on the corresponding data set. To understand which model produced the highest classification accuracy, we should take notice of the Test Acc Total column. It contains the mean measurement of that model’s performance on each data set. The best performing model for these six datasets is the Light Gradient Boosting model. It scored a mean classification accuracy of 82.8%. This was closely followed by the Convolutional Neural Network who’s mean classification accuracy is 82.4%. The worst performing model is the K-Nearest Neighbors model which scored a mean classification accuracy of 71.8%. This is a 9% reduction in accuracy when compared to the Light Gradient Boosting model.

From this experiment, we cannot determine a deep learning model can produce better classification results than a shallow learning model. We can conclude that on average, a deep learning model performs better on the six data sets used in this experiment.

# References

[1] Saad Albawi, Tareq Abed MOHAMMED, Saad AL-ZAWI, “Understanding of a Convolutional Neural Network”, Conference: The International Conference on Engineering and Technology 2017, Antalya, Turkey.

[2] Keiron O’Shea and Ryan Nash, “An Introduction to Convolutional Neural Networks”, arXiv:1511.08458(cs).

[3] <https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf>

[4] https://www.edureka.co/blog/knn-algorithm-in-r/

[5] Gongde Guo, Hui Wang, David Bell Yaxin Bi, and Kieran Greer, “KNN Model-Based Approach in Classification”, In: Meersman R., Tari Z., Schmidt D.C. (eds) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg

[6] Marques, Delano & Barradas Filho, Alex & Romariz, Alexandre & Viegas, Isabelle & Luz, Djavania & Barros Filho, Allan Kardec & Labidi, Sofiane & Ferraudo, Antonio. (2014). Recent Developments on Statistical and Neural Network Tools Focusing on Biodiesel Quality. International Journal of Computer Science and Application. 3. 97. 10.14355/ijcsa.2014.0303.01.

[7] P. J. Werbos, "Backpropagation through time: what it does and how to do it," in Proceedings of the IEEE, vol. 78, no. 10, pp. 1550-1560, Oct. 1990.

[Welcome to LightGBM's documentation! — LightGBM 2.3.2 documentation](https://lightgbm.readthedocs.io/en/latest/index.html)

[LightGBM: A Highly Efficient Gradient Boosting Decision Tree](https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf)

#### [Polish companies bankruptcy data Data Set](https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data)

[german-traffic-signs](https://www.kaggle.com/saadhaxxan/germantrafficsigns)

# APPENDIX

Table 1. Model performance on data sets

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset** | **Model** | **Test Acc** | **Val Acc** | **Train Acc** | **Execution Time (sec)** |
| Forest Cover | LGBM | 0.834 | 0.829 | 0.840 | 34.28 |
| Epileptic | LGBM | 0.974 | 0.970 | 1.000 | 1.51 |
| The Broken Machine | LGBM | 0.735 | 0.738 | 0.762 | 8.1 |
| PolishData | LGBM | 0.981 | 0.979 | 0.997 | 5.11 |
| Traffic Signs | LGBM | 0.73 | 0.71 | 1 | 3167 |
| SVHN | LGBM | 0.7135 | 0.7454 | 0.9264 | 946.16 |
| Forest Cover | KNN | 0.925 | 0.925 | 0.969 | 54.45 |
| Epileptic | KNN | 0.915 | 0.917 | 0.942 | 0.13 |
| The Broken Machine | KNN | 0.657 |  |  | 113.48 |
| Traffic Signs | KNN | 0.312 | 0.371 | 0.859 | 8446 |
| Polish Data | KNN | 0.95 | 0.95 | 0.950 | 91 |
| SVHN | KNN | 0.503 | 0.522 | 0.645 | 13788.30 |
| Forest Cover | CNN | 0.937 | 0.938 | 0.942 | 919.37 |
| Epileptic | CNN | 0.770 | 0.730 | 0.833 | 3.42 |
| The Broken Machine | CNN | 0.690 | 0.500 | 0.500 | 4076.2 |
| SVHN | CNN | 0.950 | 0.946 | 0.972 | 8147.34 |
| Polish Data | CNN | 0.623 | 0.623 | 0.716 | 250 |
| Traffic Signs | CNN | 0.977 | 0.989 | 0.992 | 87.5 |
| Traffic Signs | MLP | 0.757 | 0.721 | 0.858 | 553 |
| PolishData | MLP | 0.717 | 0.723 | 0.770 | 220.00 |
| Forest Cover | MLP | 0.854 | 0.850 | 0.856 | 485.87 |
| Epileptic | MLP | 0.947 | 0.939 | 0.954 | 10.06 |
| The Broken Machine | MLP | 0.690 | 0.694 | 0.693 | 620.73 |
| SVHN | MLP | 0.720 | 0.755 | 0.793 | 244.50 |