gr

AIN422 ASSIGNMENT 1

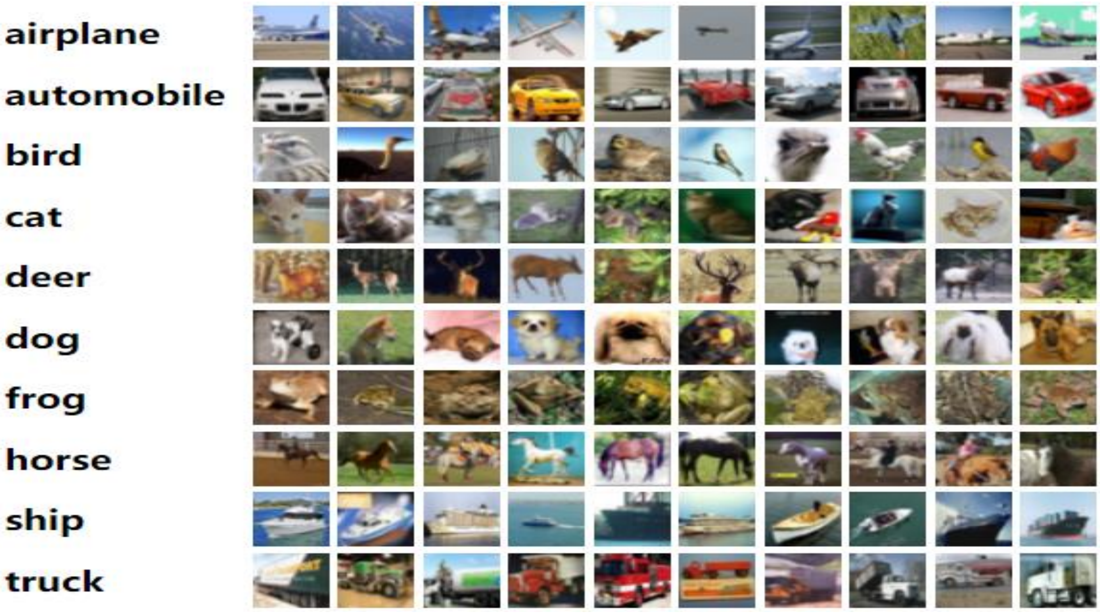
Hüseyin Yiğit Ülker
21993092
Department of AI Engineering
Hacettepe University
Ankara , Turkey
b21993092@hacettepe.edu.tr

1. Introduction

Image classification is a process of assigning a label or class to an image based on its visual content. It is a fundamental task in the field of computer vision and has numerous applications such as object recognition, facial recognition, and scene understanding. The assignment involved the implementation of three image classification algorithms: k-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Neural Network (NN). The performance evaluation of the aforementioned algorithms was conducted utilizing a pre-existing dataset, namely the Cifar-10 dataset.

2. Information about the Cifar-10 Dataset

The Cifar-10 dataset is composed of 60,000 32x32 RGB color images in 10 classes, with 6,000 images per class. It consists of a training set of 50,000 images and a test set of 10,000 images. The dataset is labeled and the classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is widely used in machine learning research as a benchmark for image classification tasks, and has contributed to the development of many deep learning architectures.



(https://paperswithcode.com/dataset/cifar-10

3. Classification Algorithms

3.1. K-Nearest Neighbor(KNN)

3.1.1. Brief Information about KNN

KNN algorithm is a non-parametric supervised machine learning algorithm used for classification and regression. In classification, the algorithm assigns a label to an input data point based on the majority class among its k nearest neighbors in the feature space. In regression, the algorithm assigns a value to an input data point based on the average value of its k nearest neighbors. The value of k is a hyperparameter that can be chosen based on the characteristics of the dataset. KNN is a memory-based algorithm, which means that it does not require training data to be explicitly modeled, but stores the entire training dataset and uses it to make predictions at runtime. The performance of KNN depends heavily on the choice of distance metric used to measure similarity between data points.

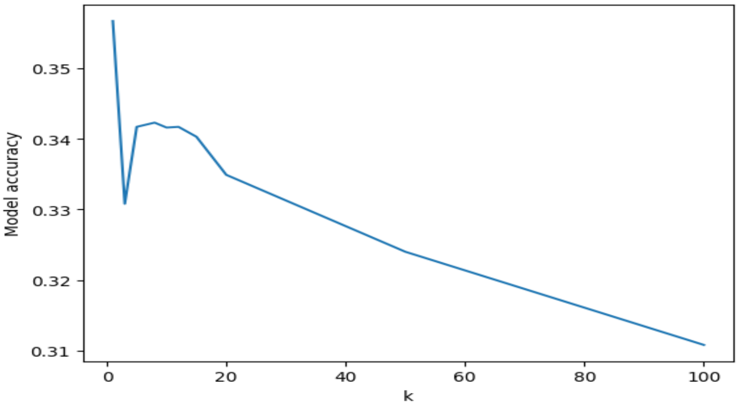
3.1.2. Experiment

In the KNN experiment section, three models were trained using different hyperparameters.

The first model was trained with the default KNN hyperparameters and a k value of 3, chosen randomly . KNN algorithm default parameters:

{'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'uniform'}.

For the second model, the KNN algorithm was used with default values, but the k value was tuned by trying out different values ranging from [1, 3, 5, 8, 10, 12, 15, 20, 50, 100], with the best accuracy achieved at k=1.



(Second model ,accuracy value according to k value)

In the third model, GridSearch algorithm was used to select the hyperparameters with the highest accuracy, based on the metrics used in the first two models. The k value for this model was restricted to [1,5,7,9,11,13,15], lower than the second model. It was observed that in the second model, as the k value increased, the accuracy value decreased.

The Grid Search algorithm was used to determine the optimal model parameters, resulting in

{'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}.

When the model was created using these parameters, the accuracy value obtained was 0.3356. However, in the second model, which was created using the entire training dataset, the accuracy value was 0.3567. To confirm that the difference in accuracy values is due to sampling of the train dataset in the third model, the second model was recreated using a sample train dataset, resulting in an accuracy value of 0.29 . This indicates that training the model using the optimal parameters determined by the GridSearch algorithm with the entire dataset would yield the highest accuracy value. Unfortunately, due to limited computational resources, this was not feasible.

Model Name	Hyperparameters	Model Description	Accuray Value
First KNN Model	{'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'uniform'}	Generated with a random k value. It is 3. Other parameters are default values. The all train dataset was used.	0.3308
Second KNN Model	{'metric': 'minkowski', 'n_neighbors': 1, 'weights': 'uniform'}	Elbow function was used while determining the k value. It is 1. Other parameters were default values. The all train data set was used.	0.3567
Third KNN Model	{'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}	All parameters were determined using the GridSearch algorithm. It was trained by taking 10000 samples from the Train dataset.	0.3356
Second KNN Model (recreated)	{'metric': 'minkowski', 'n_neighbors': 1, 'weights': 'uniform'}	Elbow function was used while determining the k value. It is 1. Other parameters were default values. It was trained by taking 10000 samples from the Train dataset.It was created to compare success with the third model.	0.2903

** In the first two models, the entire training dataset was utilized, whereas in the last model, only a subset of 10000 samples from the training dataset was used.

3.2. Support Vector Machine(SVM)

3.2.1. Brief Information about SVM

SVM algorithm is a supervised machine learning algorithm used for classification, regression, and outlier detection. In classification, the algorithm aims to find the optimal hyperplane that separates the input data points into two or more classes with the maximum margin of separation. In regression, the algorithm aims to find the optimal hyperplane that best fits the input data points. The algorithm works by mapping the input data points into a higher-dimensional feature space where a linear hyperplane can be used to separate the data points. The mapping is done using a kernel function, which can be linear or non-linear. The choice of kernel function depends on the characteristics of the dataset. SVM is a powerful algorithm for dealing with high-dimensional datasets and can handle both linearly separable and non-linearly separable data. The performance of SVM depends heavily on the choice of kernel function and the value of the regularization parameter C, which controls the trade-off between maximizing the margin and minimizing the classification error.

3.2.2. Experiment

In this section of the assignment, two SVM models were generated. In the training stage of the first model, the entire training dataset was utilized. However, due to the dataset's size and limited computational resources, the PCA algorithm was utilized to reduce the image size from 3072 (32*32*3) to 23, which reduced the dataset's dimensions. Following that, the GridSearch algorithm was used to determine the optimal parameter values for the reduced dataset. The model was trained with the parameter values

{'C': 1.0, 'gamma': 0.285, 'kernel': 'linear'}.

The first model had an accuracy of 0.371.

In the second part of the assignment, a second SVM model was created using a subset of 1000 training data instead of applying the PCA algorithm to reduce the data size. The GridSearch algorithm was then used to determine the optimal parameter values for the reduced dataset, resulting in the same parameter values as the first model:

{'C': 0.001, 'gamma': 0.1, 'kernel': 'linear'}.

The second model achieved an accuracy of 0.38, which is slightly higher than the first model.

Model Name	Hyperparameters	Model Description	Accuray Value
SVM Model_1	{'kernel': linear' 'c': 1.0, 'gammas': 0.285}	The whole training dataset and PCA algorithm were used.	0.371
SVM Model_2	{'kernel': 'linear', 'c': 0.001, 'gammas': '0.1'}	It was created with a dataset containing 1000 data obtained from the Train dataset.	0.425

3.3. Neural Network (NN)

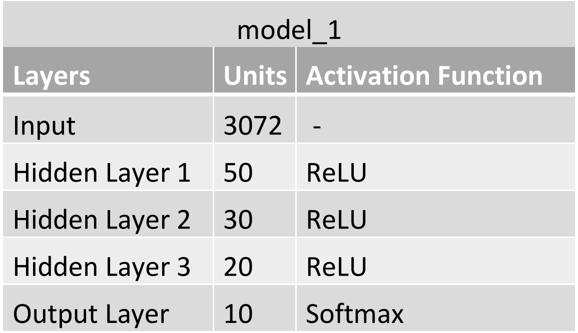
3.3.1. Brief Information about Neural Network

Neural Network algorithm is a class of supervised machine learning algorithms that is inspired by the structure and function of the human brain. NNs consist of multiple layers of interconnected artificial neurons that process and transform the input data. The input data is fed into the input layer, which passes the transformed data to the hidden layers, and finally to the output layer, which produces the predicted output. Each neuron in the network applies a non-linear activation function to the weighted sum of its inputs to introduce non-linearity and enable the network to model complex relationships in the data. The weights and biases of the neurons are learned during training using backpropagation algorithm, which iteratively adjusts the weights and biases to minimize the difference between the predicted output and the actual output. NNs are capable of learning and representing highly non-linear and complex functions, making them well-suited for a wide range of tasks, such as image recognition, speech recognition, natural language processing, and many more. The performance of NNs depends heavily on the choice of architecture, activation function, learning rate, and regularization techniques.

3.3.2. Experiment

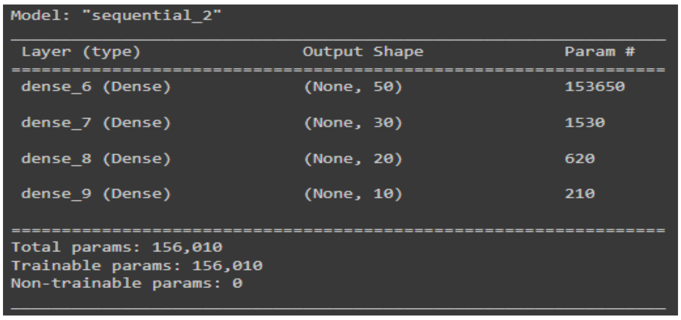
In the neural network section, which is the last part of the homework, a total of 6 neural network models were created. A baseline model was determined and each model was tried to be developed by changing different parameters.

The first model **(model_1)** was determined as the baseline model. The structure of this model is as follows:



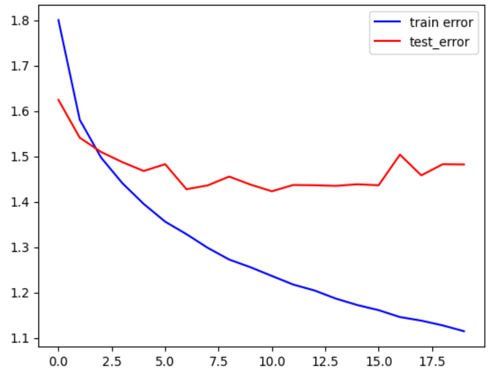
model_1		
Layers	**Units**	**Activation Function**
Input	3072	-
Hidden Layer 1	50	ReLU
Hidden Layer 2	30	ReLU
Hidden Layer 3	20	ReLU
Output Layer	10	Softmax

In addition to these, it was used SparseCategoricalCrossentropy as loss function, Adam (default learning rate = 0.001) as Optimizer and accuracy as metric.



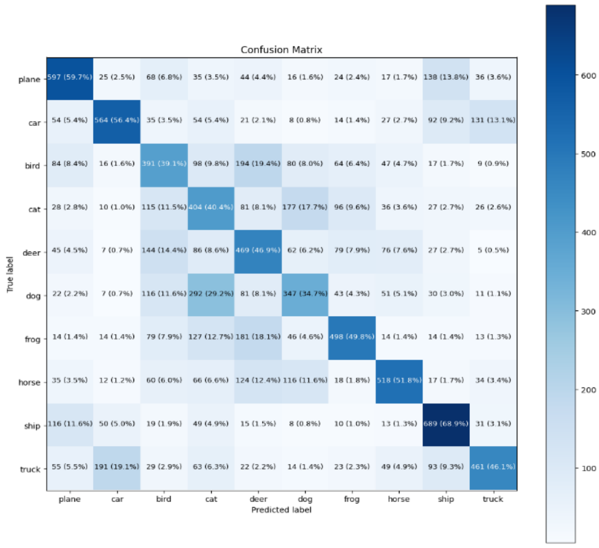
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 50)                153650

 dense_7 (Dense)             (None, 30)                1530

 dense_8 (Dense)             (None, 20)                620

 dense_9 (Dense)             (None, 10)                210

=================================================================
Total params: 156,010
Trainable params: 156,010
Non-trainable params: 0
_________________________________________________________________
```

model_1 summary

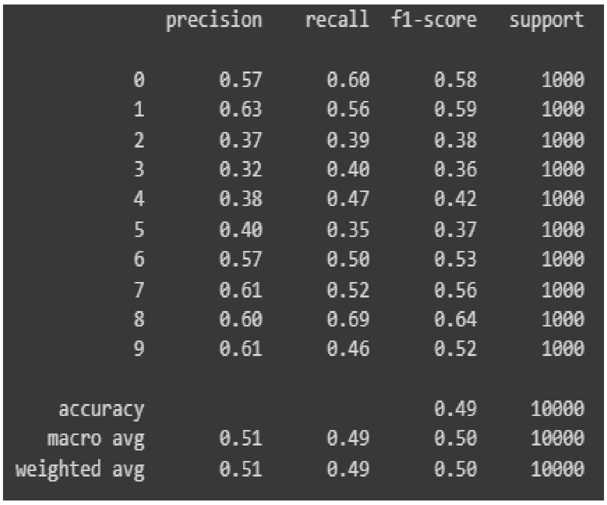


The presented graph displays the loss values of both the training and testing datasets against the number of epochs. It is essential to notice that after a particular epoch, the loss value of the test dataset becomes constant, while the loss value of the training dataset continues to decrease. This situation indicates that the model ceases to learn and starts to overfit, memorizing the training dataset instead of learning from it.

In multiclass classification, a confusion matrix is a table that shows the number of correctly and incorrectly classified examples for each class in the test data. The columns represent the predicted classes, while the rows represent the true classes. Each cell in the matrix indicates the number of instances where an actual class was predicted as a particular class. The diagonal of the matrix represents the number of correctly classified instances, while off-diagonal elements represent misclassified instances. The accuracy of the model can be calculated using the values on the diagonal. Other metrics such as precision, recall, and F1-score can also be calculated from the confusion matrix for each class. The confusion matrix provides a visual representation of how well the model is performing for each class and can be useful in identifying which classes the model is struggling to classify.



	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	597 (59.7%)	25 (2.5%)	68 (6.8%)	35 (3.5%)	44 (4.4%)	16 (1.6%)	24 (2.4%)	17 (1.7%)	138 (13.8%)	36 (3.6%)
car	54 (5.4%)	564 (56.4%)	35 (3.5%)	54 (5.4%)	21 (2.1%)	8 (0.8%)	14 (1.4%)	27 (2.7%)	92 (9.2%)	131 (13.1%)
bird	84 (8.4%)	16 (1.6%)	391 (39.1%)	98 (9.8%)	194 (19.4%)	80 (8.0%)	64 (6.4%)	47 (4.7%)	17 (1.7%)	9 (0.9%)
cat	28 (2.8%)	10 (1.0%)	115 (11.5%)	404 (40.4%)	81 (8.1%)	177 (17.7%)	96 (9.6%)	36 (3.6%)	27 (2.7%)	26 (2.6%)
deer	45 (4.5%)	7 (0.7%)	144 (14.4%)	86 (8.6%)	469 (46.9%)	62 (6.2%)	79 (7.9%)	76 (7.6%)	27 (2.7%)	5 (0.5%)
dog	22 (2.2%)	7 (0.7%)	116 (11.6%)	292 (29.2%)	81 (8.1%)	347 (34.7%)	43 (4.3%)	51 (5.1%)	30 (3.0%)	11 (1.1%)
frog	14 (1.4%)	14 (1.4%)	79 (7.9%)	127 (12.7%)	181 (18.1%)	46 (4.6%)	498 (49.8%)	14 (1.4%)	14 (1.4%)	13 (1.3%)
horse	35 (3.5%)	12 (1.2%)	60 (6.0%)	66 (6.6%)	124 (12.4%)	116 (11.6%)	18 (1.8%)	518 (51.8%)	17 (1.7%)	34 (3.4%)
ship	116 (11.6%)	50 (5.0%)	19 (1.9%)	49 (4.9%)	15 (1.5%)	8 (0.8%)	10 (1.0%)	13 (1.3%)	689 (68.9%)	31 (3.1%)
truck	55 (5.5%)	191 (19.1%)	29 (2.9%)	63 (6.3%)	22 (2.2%)	14 (1.4%)	23 (2.3%)	49 (4.9%)	93 (9.3%)	461 (46.1%)

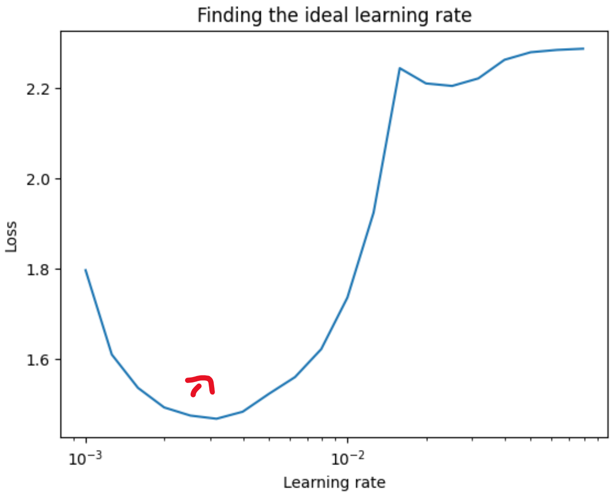


```
              precision    recall  f1-score   support

           0       0.57      0.60      0.58      1000
           1       0.63      0.56      0.59      1000
           2       0.37      0.39      0.38      1000
           3       0.32      0.40      0.36      1000
           4       0.38      0.47      0.42      1000
           5       0.40      0.35      0.37      1000
           6       0.57      0.50      0.53      1000
           7       0.61      0.52      0.56      1000
           8       0.60      0.69      0.64      1000
           9       0.61      0.46      0.52      1000

    accuracy                           0.49     10000
   macro avg       0.51      0.49      0.50     10000
weighted avg       0.51      0.49      0.50     10000
```

model_1 Confusion Matrix

model_1 evaluation scores

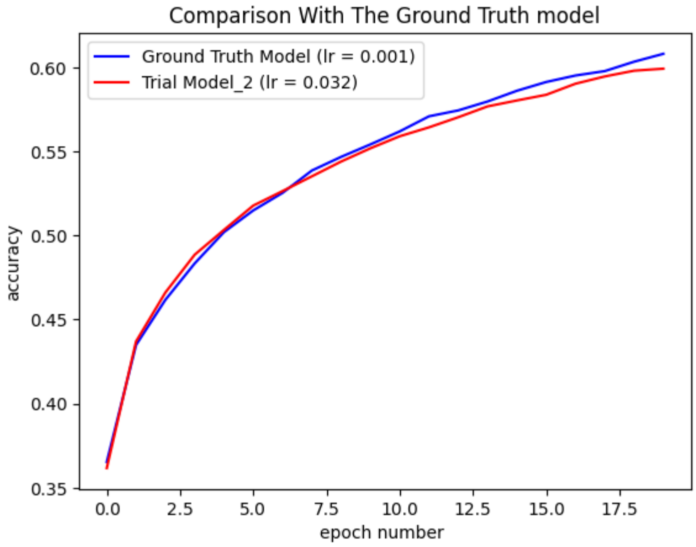
In **model_2**, the aim was to determine the optimal value for the learning rate, which was set to the default value in model_1. It was concluded that using a value of 0.032 instead of the default value of 0.001 would result in better performance.



To figure out the ideal value of the learning rate (at least the ideal value to *begin* training our model), the rule of thumb is to take the learning rate value where the loss is still decreasing but not quite flattened out (usually about 10x smaller than the bottom of the curve).

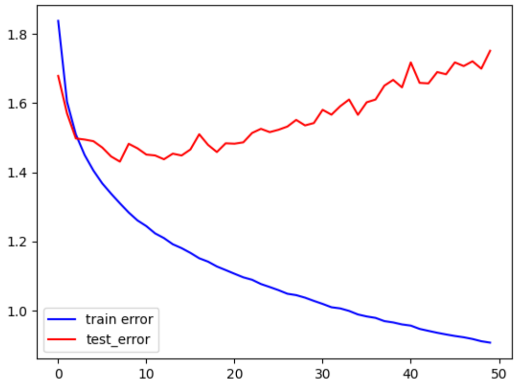
In this case, our ideal learning rate is 0.32

Note: Adam optimizer is used in the model.



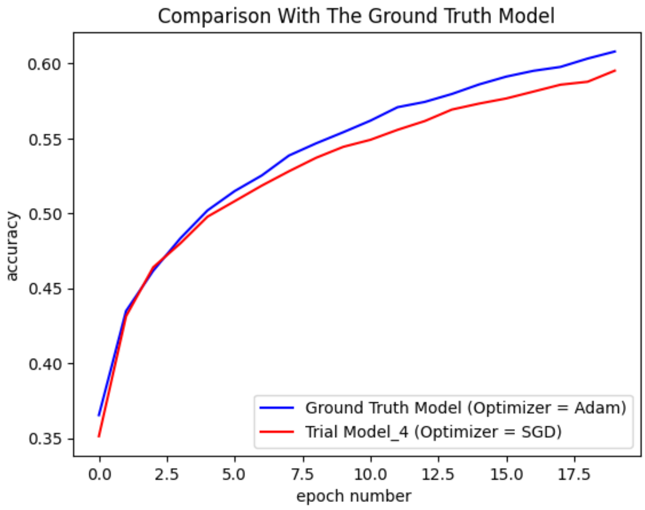
After modifying the learning rate and creating the model, the accuracy slightly decreased. When examining the graph, it was observed that model_2 provides better results at lower epoch values, whereas model_1 gives better accuracy as the number of epochs increases.

In **model_3**, the impact of the number of epochs was investigated. The number of epochs in model_1, which was 20, was increased to 50 epochs in model_3, and the resulting changes in the model's performance were evaluated.



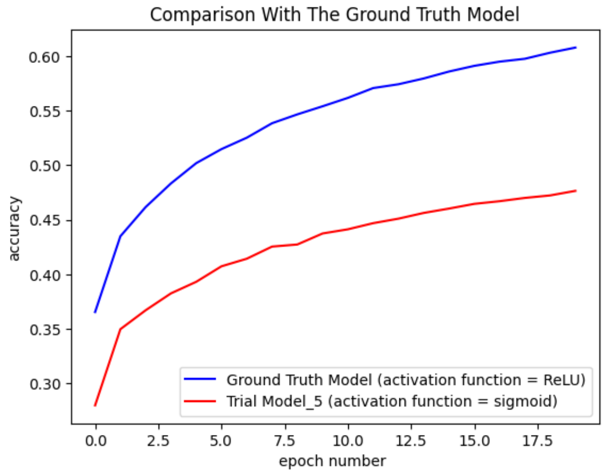
As shown in the graph, increasing the number of epochs leads the model to overfit, where the train error decreases while the test error increases. This indicates that the model is losing its generalization capability and instead starts to memorize the training dataset. Consequently, the overall performance of the model decreases. In the case of model_3, the train accuracy value of 0.6080 in model_1 decreased to 0.6696 as the number of epochs increased.

In **model_4**, the effect of the optimizer on the success of the model was examined. Instead of the adam optimizer used in model_1, the stochastic gradient descent (SGD) optimizer was used in model_4.



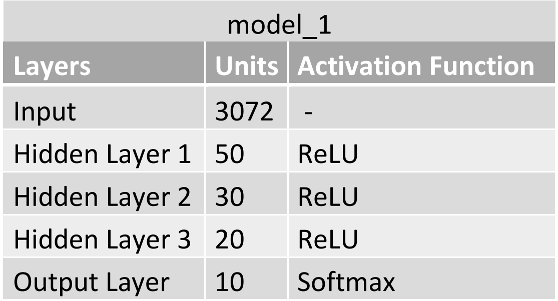
According to the graph, it can be observed that the performance of model_4, which uses the SGD optimizer, was not as good as model_1 that uses the Adam optimizer, since it showed slower performance . As a result, the train accuracy score of model_4 was lower than that of model_1, where it dropped from 0.6080 to 0.5952 .

In the **model_5** , the impact of changing the activation function was investigated. Specifically, instead of using the rectified linear unit (ReLU) activation function in the hidden layers as in model_1, the sigmoid activation function was employed, and the performance of the model was evaluated accordingly.



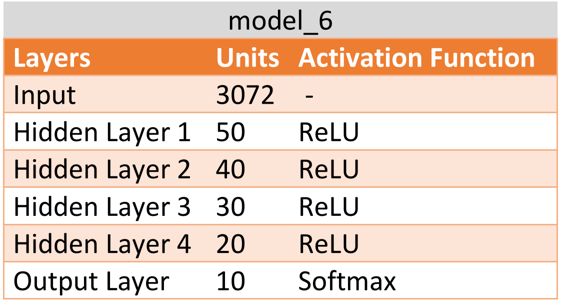
Model_5, which used the sigmoid activation function instead of ReLU in the hidden layers, exhibited a significantly lower level of performance than Model_1. This is attributed to the "Vanishing Gradient" issue caused by the sigmoid function, which results in a lack of learning and has a negative impact on the overall success of the model. As a result, the training accuracy of Model_5 dropped from 0.6080 to 0.4765.

In the last model (**model_6**), we investigated the impact of the model structure on its success. We created model_6 by adding an extra hidden layer to the architecture of model_1.



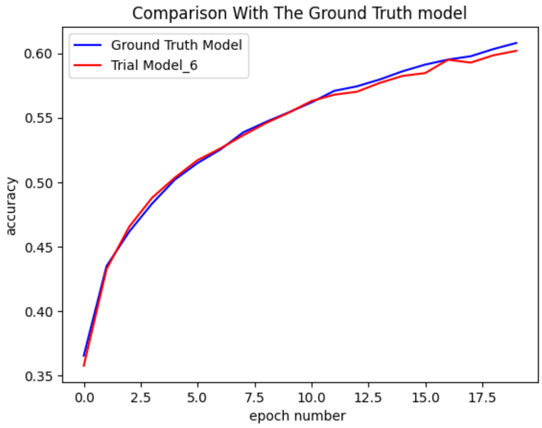
model_1

Layers	Units	Activation Function
Input	3072	-
Hidden Layer 1	50	ReLU
Hidden Layer 2	30	ReLU
Hidden Layer 3	20	ReLU
Output Layer	10	Softmax



model_6

Layers	Units	Activation Function
Input	3072	-
Hidden Layer 1	50	ReLU
Hidden Layer 2	40	ReLU
Hidden Layer 3	30	ReLU
Hidden Layer 4	20	ReLU
Output Layer	10	Softmax



Based on the graph, it can be observed that there was not a significant difference in the training success between model_1 and model_6. However, model_1 performed better in terms of overall success. The train accuracy value of model_1 was 0.4937, while the train accuracy value of model_6 was slightly lower at 0.4819.

4. Comparison Algorithms

Model Name	**Test Accuracy Value**
KNN model_1	0.3308
KNN model_2	0.3567
KNN model_3	0.3356
KNN model_4	0.2903
SVM model_1	0.3308
SVM model_2	0.425
NN model_1	0.4937
NN model_2	0.4993
NN model_3	0.4780
NN model_4	0.4790
NN model_5	0.4426
NN model_6	0.4947

This table displays the test accuracy values of various machine learning models trained on a specific dataset. The models include KNN, SVM, and NN models with different hyperparameters.

The results indicate that changing the hyperparameters can significantly affect the accuracy of the model. For instance, the accuracy of KNN model_2 is higher than KNN model_1, while the accuracy of KNN model_4 is the lowest among all KNN models. Similarly, SVM model_2 has higher accuracy than SVM model_1. Among NN models, NN model_2 has the highest accuracy, while NN model_5 has the lowest accuracy.

Therefore, it is crucial to select appropriate hyperparameters that optimize the performance of the model. Hyperparameter tuning can be performed using techniques like grid search, random search, or Bayesian optimization to find the best set of hyperparameters that maximize the accuracy of the model.