Artificial Neural Networks – Final Report

Efe Mert Çırpar - 040190247

Electronics and Communication Engineering - Faculty of Electrical and Electronics

Important Remark

All algorithms must run separately since I am adjusting train-test arrays according to my CPU and my RAM. For plotting, I used optimal parameters.

Introduction

CIFAR-10 is a dataset of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. It is widely used for machine learning and computer vision research and has been a popular dataset for benchmarking image classification models.

Data Preprocessing

Data preprocessing is an essential step in machine learning, and this is especially true when working with image datasets like CIFAR-10. One way to easily access and load the CIFAR-10 dataset is by using the "keras.datasets" module, which is included in the Keras library. This module allows you to easily load the CIFAR-10 dataset by calling the "cifar10.load_data()" function. With following code lines from my main.py file I created X_train, X_test, Y_train, and Y_test arrays and shaped them according to my code in order to use them in Machine Learning Algorithms.

```
import keras as k
from keras.datasets import cifar10
#Loading Dataset
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
X_train = X_train.reshape(50000,3072)
X_test = X_test.reshape(10000,3072)
```

Methods

1) KNN

My KNN algorithm starts with getting KNeighborsClassifier() function and hyperparameters. My first parameter was "n_neighbor" numbers which I chose 1,3,5 (increasing with 2) because at the beginning I chose different numbers and at the end 5 was the best parameter all the time so I chose these numbers to make things easier for my computer. And as my second parameter, I chose "weights" and since it's a more straightforward parameter I used uniform and distance types. In the end, K=5 and Weight = Distance were the best fit according to my python code. Then used plotting algorithms like homework 4.

<u>Fitting Code Lines:</u>

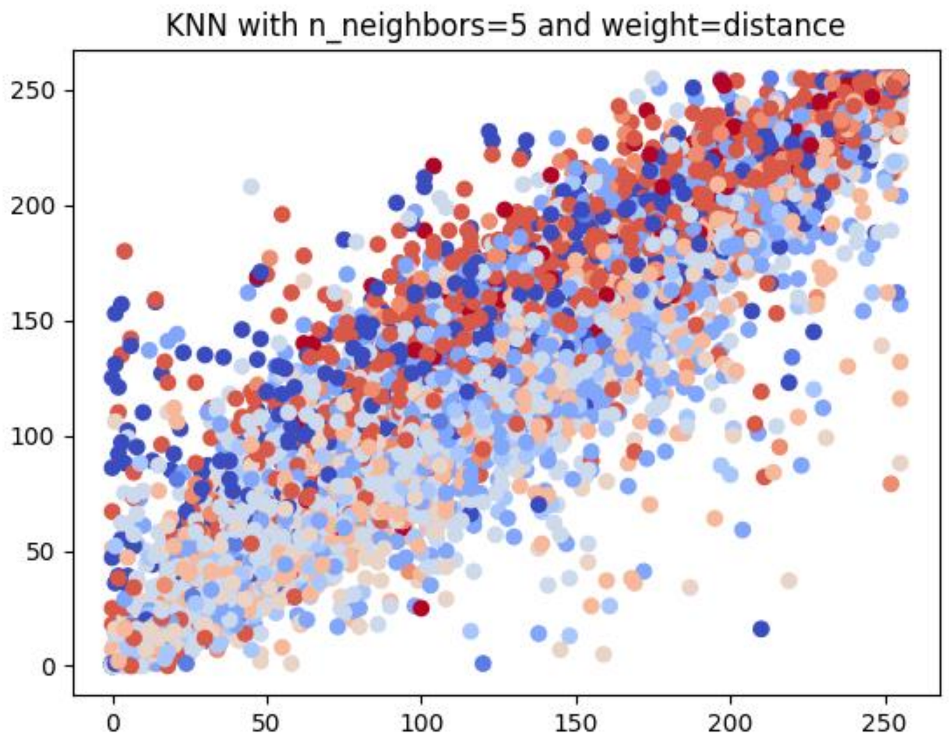
```
knn_hptuning_grid = {'n_neighbors': [1, 3, 5], 'weights': ['uniform',
'distance']}
knn_gridsearchcv = GridSearchCV(knn, knn_hptuning_grid, cv=5)
knn_gridsearchcv.fit(X_train, Y_train)
```

<u>Output:</u>

Best Parameters: {'n_neighbors': 5, 'weights': 'distance'}

Test Set Accuracy: 0.3567

<u>Plotting (With Best Fitted Parameters):</u>



2) Linear Regression

Like my other algorithms, my linear regression algorithm started with getting LinearRegression() and hyperparameters. In this algorithm, I only used the “fit_intercept” parameter and tried to get “True” or “False” values from it. As my result fit_intercept = True was the best fit according to my python code. As an accuracy score, since Linear Regression is not running the accuracy_score line for continuous data, I found MSE(mean squared error) for my data. After that, I plotted algorithms.

Fitting Code Lines:

```
lr_hptuning_grid = {'fit_intercept': [True, False]}
#Performing Hyperparameter and Finding The Best Parameters
lr_gridsearchcv = GridSearchCV(lr, lr_hptuning_grid, cv=5, n_jobs=-1)
lr_gridsearchcv.fit(X_train, Y_train)
#Fittinng Best Hyperparameter
print("Best Hyperparameters:", lr_gridsearchcv.best_params_)
lr_best_model = lr_gridsearchcv.best_estimator_
lr_best_model.fit(X_train, Y_train)
```

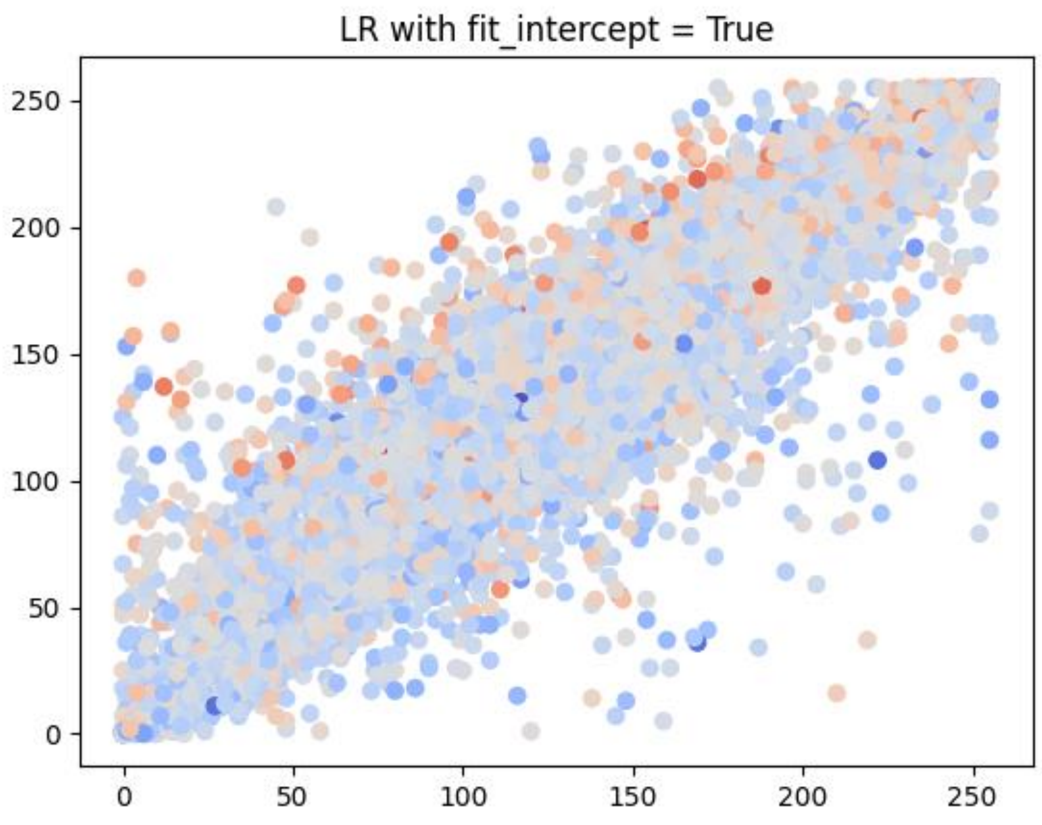
MSE:

```
mse = mean_squared_error(Y_test, Y_pred)
print("Mean Squared Error:", mse)
```

Output:

```
Best Hyperparameters: {'fit_intercept': True}
Mean Squared Error: 8.03262844577586
```

Plotting (With Best Fitted Parameters):



---

3) SVM

At this part training was not ending so I needed to reduce and scale some of my data due to my computer's deficiencies. After that, I chose my parameters as "C" and "Kernel" functions. I chose the C parameter since it is a substantial parameter for error penalty. Another parameter, kernel, is a parameter to determine the shape of the function. As output, I got C =10 and Kernel = RBF.

Data Adjustment Lines:

```
#Scale The Data with MinMaxScaler or StandardScaler
scaler = MinMaxScaler()
#scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
#Using Some Part of The Data Because My Computer Doesn't Allow To Run All
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
Y_train = pd.DataFrame(Y_train)
Y_test = pd.DataFrame(Y_test)
X_train = X_train.iloc[0:2000,:]
Y_train = Y_train.iloc[0:2000,:]
X_test = X_test.iloc[0:400,:]
Y_test = Y_test.iloc[0:400,:]
```

Fitting Code Lines:

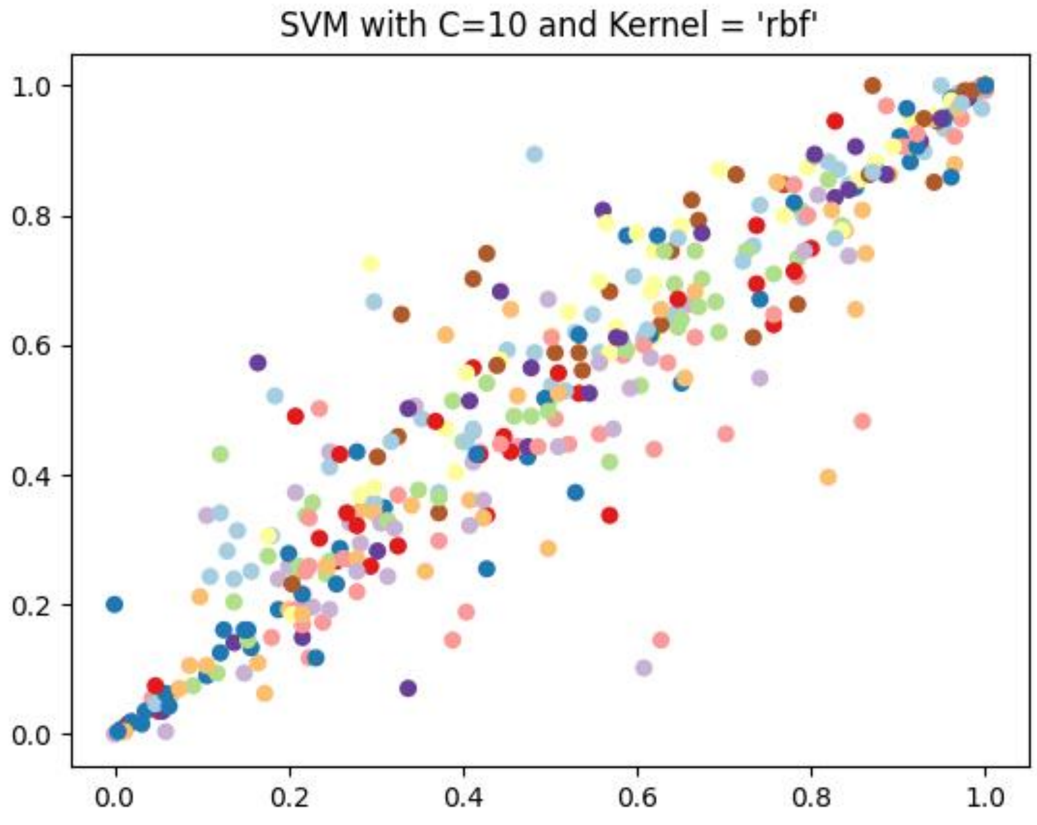
```
#Defining SVM and Its Parameter Grid for Hyperparameter Tuning
svm = SVC()
svm_hptuning_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear', 'rbf']}
#Performing Hyperparameter and Finding The Best Parameters
svm_gridsearchcv = GridSearchCV(svm, svm_hptuning_grid, cv=5)
svm_gridsearchcv.fit(X_train, Y_train)
```

Output:

Best Hyperparameters: {'C': 10, 'kernel': 'rbf'}

Test Set Accuracy: 0.3625

<u>Plotting (With Best Fitted Parameters):</u>



---

4) MLP

With 100 maximum iterations, I used hidden_layer, learning_rate, and alpha parameters to create my hyperparameter. I chose hidden layers to determine the number of more optimal hidden layers in this code. Learning rate and alpha were also common parameters for this algorithm. In the end, hidden_layer_size = 5, learning_rate = constant, and alpha = 0.0001 were the optimal results.

Fitting Code Lines:

```
mlp = MLPClassifier(max_iter=100, random_state=1)
mlp_hptuning_grid = {'hidden_layer_sizes': [(5,), (10,)], 'alpha': [0.0001,
0.05], 'learning_rate': ['constant','adaptive'],}
#Performing Hyperparameter and Finding The Best Parameters
mlp_gridsearchcv = GridSearchCV(mlp, mlp_hptuning_grid, cv=5)
mlp_gridsearchcv.fit(X_train, Y_train)
```

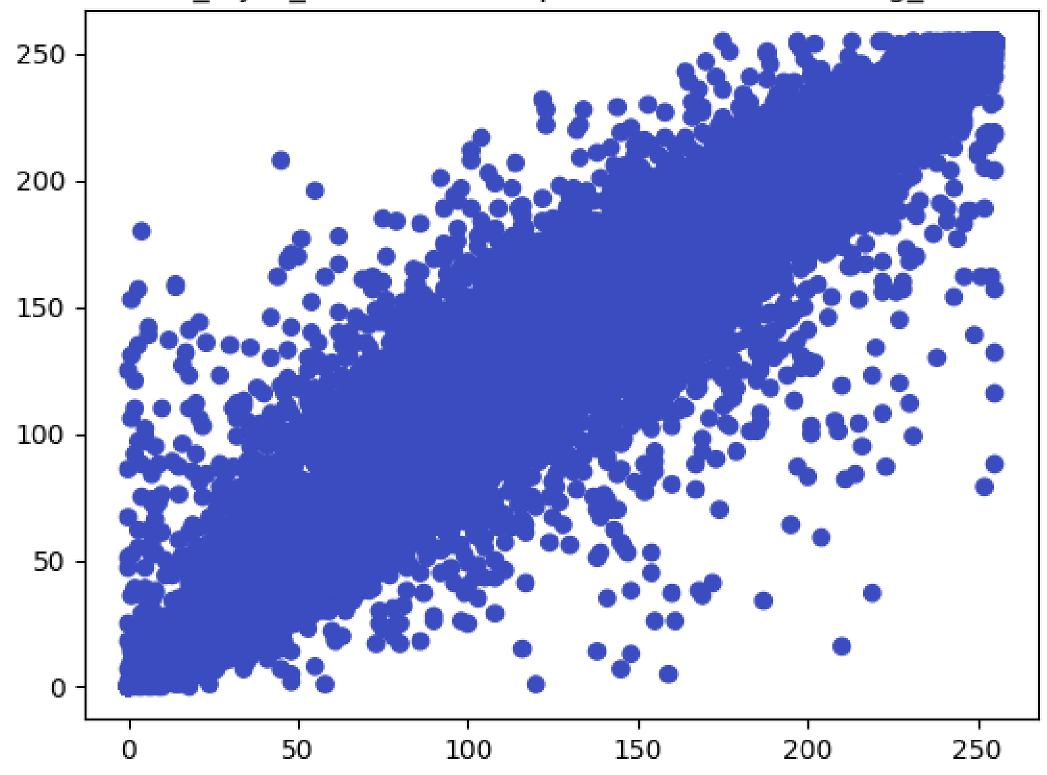
Output:

```
Best Hyperparameters: {'alpha': 0.0001, 'hidden_layer_sizes': (5,), 'learning_rate': 'constant'}

Test Set Accuracy:  0.1
```

Plotting (With Best Fitted Parameters):

MLP with hidden_layer_size = (5,0), alpha = 0.0001, learning_rate = 'constar



5) CNN

In the beginning, I normalized my data with /=255 line. Then, I convert Y to_categorical(10) since there are 10 classes. This model uses 32 and 64 filters with (4,4) kernel size. As an activation parameter, I used “relu” and as padding parameter, I used “same”. With these parameters, I plotted accuracy/epoch and loss/epoch plots. I chose epoch = 10, because of the training time and my computer.

Fitting Code Lines:

```
#CNN Architecture
cnn = Sequential()
cnn.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))
cnn.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10, activation='softmax'))
#Compiling CNN
cnn.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.0001,
decay=1e-6), metrics=['accuracy'])
#Training CNN
history = cnn.fit(X_train, Y_train, batch_size=32, epochs=10,
validation_data=(X_test, Y_test))
```

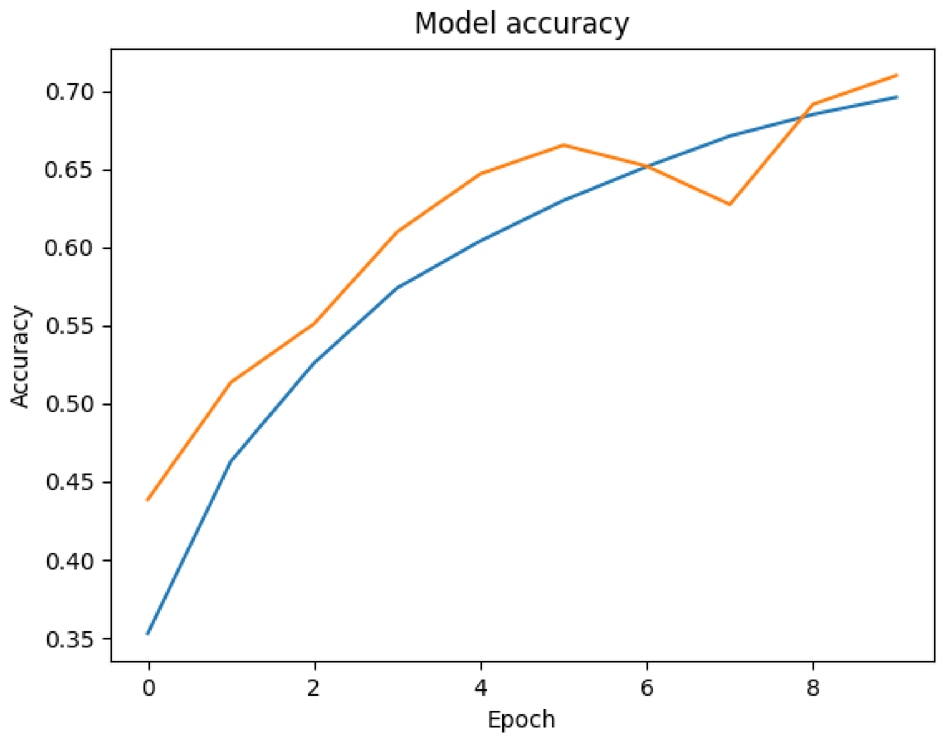
Output:

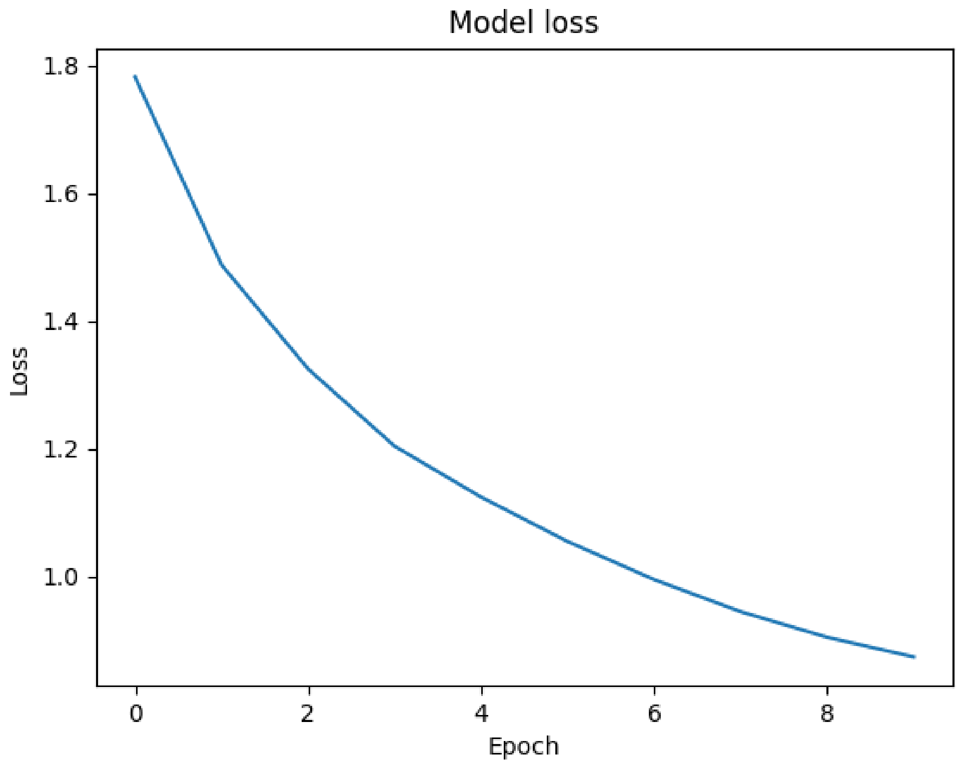
```
accuracy: 0.6959 - val_loss: 0.8386 - val_accuracy: 0.7098

Test loss: 0.8385558724403381

Test accuracy: 0.7098000049591064
```

Plotting:


Model accuracy
Accuracy
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2
4
6
8


Model loss
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0
2
4
6
8

Results Model

Method	**Output Type**	**Output**
KNN	Accuracy Score	0.3567
LR	Mean Squared Error	8.0326
SVM	Accuracy Score	0.3625
MLP	Accuracy Score	0.1
CNN	Accuracy Score	0. 7098

Discussion

According to these estimations, as expected, CNN was the best-fitted algorithm compared to KNN, LR, SVM (only 2000 training inputs were used), and MLP with a remarkable difference in accuracy.

Conclusion

I learned how to code these algorithms in a more efficient way, how to use online sources and how we should react to some specific errors in this assignment.

I might have changed parameters/epoch etc. to get more accuracy if there were no technical restrictions. I believe in better devices these algorithms can be working in a more accurate way.