Image Classification on CIFAR-10 using Machine Learning and Deep Learning Models

COMP 472: Artificial Intelligence

Concordia University

November 25th, 2024

Rayan Alkayal - 40051210

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

Table of Contents

Table of Contents **2**
1. Abstract **3**
2. Dataset Overview **4**
2.1 Load Dataset 4
2.2 Feature Extraction 4
3. AI Models **5**
3.1 Gaussian Naive Bayes 5
3.2 Decision Tree 6
3.2.1 Decision tree with depth = 10 6
3.2.2 Decision tree with depth = 25 7
3.2.3 Decision tree with depth = 50 8
3.2.4 Summary 9
3.3 Multi-Layer Perceptron (MLP) 10
3.3.1 Shortened MLP 10
3.3.2 Base MLP 10
3.3.3 Extended MLP 11
3.3.4 Summary 11
3.4 Convolutional Neural Network (CNN) 13
3.4.1 Varying Convolutional Layers 13
3.4.2 Kernel Size = 3 13
3.4.3 Kernel Size = 5 14
3.4.4 Summary 14
4. References **16**

1. Abstract

This report explores image classification on the CIFAR-10 dataset using four AI models: Naive Bayes, Decision Tree, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN). The goal is to compare and evaluate the effectiveness of these models in terms of accuracy, precision, recall, and F1-score. Features were extracted using a pre-trained ResNet-18 CNN and reduced to 50 dimensions using Principal Component Analysis (PCA) for the Naive Bayes, Decision Tree, and MLP implementations. For CNN, model A of the ConvNet configuration was used, adapted for CIFAR-10's input size and evaluated with different depths and kernel sizes. Experiments involved varying model depth, layer sizes, and kernel sizes to analyze their impact on the models' ability to learn and generalize. Results show that deeper architectures like CNNs achieved superior accuracy, while simpler models like Naive Bayes and Decision Trees demonstrated limitations in handling complex data distributions. This report provides insights into the trade-offs between computational complexity and model performance, offering a comprehensive comparison of these approaches.

2. Dataset Overview

2.1 Load Dataset

In order to classify images using AI models, the first task is to download the full dataset of images from Torchvision datasets [1] and ingest them using ResNet-18 for feature extraction. To do so, all images are resized to 224 x 224 to match the input size expected by the model, and each pixel is also normalized to have a mean and standard deviation of 0.5. Next, the dataset is split into training and test sets, and the data is partitioned using a for loop to only include the first 500 training images and 100 test images per class. Training and testing datasets are then loaded into memory using PyTorch's DataLoader, which facilitates batching and shuffling of data for model training [2]. Finally, the counts of the training and test images are printed to the terminal to confirm that the dataset was loaded successfully.

2.2 Feature Extraction

The CIFAR-10 dataset undergoes preprocessing to ensure that it is compatible with a pre-trained ResNet-18 model (with the final layer removed) from Torchvision [3]. Feature extraction is performed to generate 512-dimensional feature vectors from the second to last layer. This method captures high level image features and reduces the data's dimensionality, yet still retains critical information for accurate classification [4].

Features are generated for both the training and test sets, and for each dataset, the feature vectors and corresponding labels are saved in .pt format for PyTorch compatibility, and .npy format for NumPy compatibility. The resulting feature matrices have the following shapes:

- Training Set: [5000, 512] (5000 images, each represented by a 512-dimensional vector)
- Test Set: [1000, 512] (1000 images, each represented similarly)

3. AI Models

3.1 Gaussian Naive Bayes

Before implementing the GNB algorithm in Python, some research had to be done on forming predictions for the outcomes of the test set. DataCamp provided a clear explanation of implementing the Naive Bayes algorithm using the Scikit-learn library, as well as a guide for interpreting metrics like accuracy and confusion matrices, which are critical in evaluating the performances [5].

Two implementations (custom and Scikit-learn) of the Gaussian Naive Bayes algorithm were developed in Python and were fitted on the training feature vectors of all 10 classes. Both implementations ended up producing identical results:

```
Confusion Matrix:
[[66  0  2  3  3  0  0  0 23  3]
 [ 2 78  0  2  0  0  1  0  5 12]
 [ 6  0 52  4 15  7 15  1  0  0]
 [ 1  0  8 63  6 14  5  3  0  0]
 [ 2  0  2  6 75  1  1 11  1  1]
 [ 0  0  4 17  6 67  4  1  1  0]
 [ 1  0  4  3 12  2 77  0  1  0]
 [ 1  2  1  8 11  2  0 72  1  2]
 [ 7  0  0  0  2  0  0  0 85  6]
 [ 3  6  0  0  1  0  0  2  5 83]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.66      0.70       100
           1       0.91      0.78      0.84       100
           2       0.71      0.52      0.60       100
           3       0.59      0.63      0.61       100
           4       0.57      0.75      0.65       100
           5       0.72      0.67      0.69       100
           6       0.75      0.77      0.76       100
           7       0.80      0.72      0.76       100
           8       0.70      0.85      0.77       100
           9       0.78      0.83      0.80       100

    accuracy                           0.72      1000
   macro avg       0.73      0.72      0.72      1000
weighted avg       0.73      0.72      0.72      1000
```

The rows in the confusion matrix represent true classes, and the columns correspond to the predicted classes. An entry at any position (i, j) represents the number of samples from class i that were predicted as class j. Entries are indexed from 0-9 on each axis, starting at the top left (0,0) in the confusion matrix above. It can be interpreted that class 8 (ship) shows the strongest performance with 85% recall. The model demonstrated strong precision and recall for classes like 1 (automobile), 6 (frog), and 8 (ship), while classes like 2 (bird) and 3 (cat) had more instances of misclassification.

3.2 Decision Tree

The decision tree classifier was evaluated on the CIFAR-10 dataset using custom and Scikit-learn implementations. The model's performance was analyzed at three depths: 10, 25, and 50 (maximum depth), and the results from both implementations this time were not identical but close. Increasing the depth of the decision tree from 10 to 50 added about a minute to the training process, which is not worth it considering that the results showed a decrease in performance after 10 layers.

3.2.1 Decision tree with depth = 10

Below are the confusion matrix and classification report for the custom implementation of the decision tree, which had an accuracy of 48.20%.

Confusion Matrix:

```
[[62  3  5  7  7  1  3  1  9  2]
 [ 9 54  1  8  1  1  1  0 11 14]
 [ 8  3 35 15 12  6 14  6  0  1]
 [ 3  7  8 50  7 12  9  2  2  0]
 [ 3  2 13  6 51  3  6 15  0  1]
 [ 5  1  6 30  7 40  5  4  1  1]
 [ 1  4 14 18  9  5 43  2  1  3]
 [10  5  2 13 13  6  4 44  1  2]
 [25  7  1  1  3  0  0  3 51  9]
 [ 6 22  1  5  1  1  4  2  6 52]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.47	0.62	0.53	100
1	0.50	0.54	0.52	100
2	0.41	0.35	0.38	100
3	0.33	0.50	0.40	100
4	0.46	0.51	0.48	100
5	0.53	0.40	0.46	100
6	0.48	0.43	0.46	100
7	0.56	0.44	0.49	100
8	0.62	0.51	0.56	100
9	0.61	0.52	0.56	100
accuracy			0.48	1000
macro avg	0.50	0.48	0.48	1000
weighted avg	0.50	0.48	0.48	1000

Below are the confusion matrix and classification report for the Scikit-learn implementation of the decision tree, which had an accuracy of 47.20%.

Confusion Matrix:

```
[[64  4  4  6  6  1  2  1 10  2]
 [10 52  1  7  1  1  1  1 12 14]
 [ 9  3 33 14 15  6 15  5  0  0]
 [ 3  6  9 46  7 14  9  3  3  0]
 [ 3  2 13  8 50  4  5 15  0  0]
 [ 4  2  7 29 10 39  3  4  1  1]
 [ 2  4 11 18  9  5 44  2  3  2]
 [10  7  5 13 11  8  3 41  1  1]
 [29  7  1  2  4  0  0  2 50  5]
 [ 7 22  0  6  2  1  2  2  5 53]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.45	0.64	0.53	100
1	0.48	0.52	0.50	100
2	0.39	0.33	0.36	100
3	0.31	0.46	0.37	100
4	0.43	0.50	0.47	100
5	0.49	0.39	0.44	100
6	0.52	0.44	0.48	100
7	0.54	0.41	0.47	100
8	0.59	0.50	0.54	100
9	0.68	0.53	0.60	100
accuracy			0.47	1000
macro avg	0.49	0.47	0.47	1000
weighted avg	0.49	0.47	0.47	1000

3.2.2 Decision tree with depth = 25

Below are the confusion matrix and classification report for the custom implementation of the decision tree, which had an accuracy of 45.80%.

Confusion Matrix:

```
[[54  5  6  7  6  2  2  1 11  6]
 [ 4 53  2  4  0  5  2  0 12 18]
 [ 7  2 34 11 10  9 17  7  0  3]
 [ 3  7  8 38  8 19 10  2  2  3]
 [ 4  1 14  7 38  8  7 17  1  3]
 [ 6  0  6 15  6 47 10  6  2  2]
 [ 5  5  7 14  9  8 47  2  0  3]
 [ 9  3  6  8 14  9  4 42  2  3]
 [19  1  3  4  2  1  1  2 52 15]
 [ 5 19  2  4  0  4  3  3  7 53]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.47	0.54	0.50	100
1	0.55	0.53	0.54	100
2	0.39	0.34	0.36	100
3	0.34	0.38	0.36	100
4	0.41	0.38	0.39	100
5	0.42	0.47	0.44	100
6	0.46	0.47	0.46	100
7	0.51	0.42	0.46	100
8	0.58	0.52	0.55	100
9	0.49	0.53	0.51	100
accuracy			0.46	1000
macro avg	0.46	0.46	0.46	1000
weighted avg	0.46	0.46	0.46	1000

Below are the confusion matrix and classification report for the Scikit-learn implementation of the decision tree, which had an accuracy of 45.70%.

Confusion Matrix:

```
[[60  4  9  4  7  2  0  1  7  6]
 [ 9 50  2  2  0  3  2  0 14 18]
 [10  1 32 12 10  9 19  6  0  1]
 [ 5  4  6 35 12 15 11  8  3  1]
 [ 3  1 17 12 39  4  8 13  0  3]
 [ 4  0  5 18 10 54  3  2  1  3]
 [ 4  3 10 14 13  5 45  2  2  2]
 [ 9  3 11  9 14  4  1 41  3  5]
 [17  3  3  3  7  1  1  1 49 15]
 [ 6 22  1  6  0  4  2  1  6 52]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.47	0.60	0.53	100
1	0.55	0.50	0.52	100
2	0.33	0.32	0.33	100
3	0.30	0.35	0.33	100
4	0.35	0.39	0.37	100
5	0.53	0.54	0.54	100
6	0.49	0.45	0.47	100
7	0.55	0.41	0.47	100
8	0.58	0.49	0.53	100
9	0.49	0.52	0.50	100
accuracy			0.46	1000
macro avg	0.46	0.46	0.46	1000
weighted avg	0.46	0.46	0.46	1000

3.2.3 Decision tree with depth = 50

Below are the confusion matrix and classification report for the custom implementation of the decision tree, which had an accuracy of 45.80%.

Confusion Matrix:

```
[[54  5  6  7  6  2  2  1 11  6]
 [ 4 53  2  4  0  5  2  0 12 18]
 [ 7  2 34 11 10  9 17  7  0  3]
 [ 3  7  8 38  8 19 10  2  2  3]
 [ 4  1 14  7 38  8  7 17  1  3]
 [ 6  0  6 15  6 47 10  6  2  2]
 [ 5  5  7 14  9  8 47  2  0  3]
 [ 9  3  6  8 14  9  4 42  2  3]
 [19  1  3  4  2  1  1  2 52 15]
 [ 5 19  2  4  0  4  3  3  7 53]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.47	0.54	0.50	100
1	0.55	0.53	0.54	100
2	0.39	0.34	0.36	100
3	0.34	0.38	0.36	100
4	0.41	0.38	0.39	100
5	0.42	0.47	0.44	100
6	0.46	0.47	0.46	100
7	0.51	0.42	0.46	100
8	0.58	0.52	0.55	100
9	0.49	0.53	0.51	100
accuracy			0.46	1000
macro avg	0.46	0.46	0.46	1000
weighted avg	0.46	0.46	0.46	1000

Below are the confusion matrix and classification report for the Scikit-learn implementation of the decision tree, which had an accuracy of 46.60%

Confusion Matrix:

```
[[57  3  4  9  8  2  1  0 11  5]
 [ 8 53  2  2  1  3  4  1 10 16]
 [ 9  2 33 12  8  8 17  9  1  1]
 [ 4  3  5 42 14 16  8  3  3  2]
 [ 4  2 13  8 38  8 12 13  0  2]
 [ 3  3  6 17  8 51  4  4  2  2]
 [ 3  3 10 15 10  5 46  2  4  2]
 [ 9  5  5  9 15  7  3 41  2  4]
 [18  6  2  6  4  1  0  2 53  8]
 [ 6 21  2  5  0  2  3  3  6 52]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.47	0.57	0.52	100
1	0.52	0.53	0.53	100
2	0.40	0.33	0.36	100
3	0.34	0.42	0.37	100
4	0.36	0.38	0.37	100
5	0.50	0.51	0.50	100
6	0.47	0.46	0.46	100
7	0.53	0.41	0.46	100
8	0.58	0.53	0.55	100
9	0.55	0.52	0.54	100
accuracy			0.47	1000
macro avg	0.47	0.47	0.47	1000
weighted avg	0.47	0.47	0.47	1000

3.2.4 Summary

The training methodology for the decision tree implementation involves using a maximum_depth parameter to control the tree's complexity. The custom decision tree employs a recursive splitting approach based on minimizing Gini impurity, as seen below:

```
gini = self._gini_index(y[left_idx], y[right_idx])
if gini < best_gini:
   best_gini = gini
   best_feature = feature_idx
   best_threshold = threshold
```

The model stops training once the maximum_depth is reached or other base conditions such as all samples belonging to one class are met. For the Scikit-learn decision tree, the DecisionTreeClassifier is used, which implements Gini impurity by default [6].

The dataset is processed in memory without explicitly mini-batching, as decision trees are non-parametric models that fit the data by splitting recursively rather than updating weights iteratively. Optimization techniques like early stopping, pruning, or regularization are not present in this implementation, leaving tree complexity entirely depth-dependent.

The custom decision tree implementation with depth 10 achieves the highest accuracy at 48.20%, while depths 25 and 50 yield lower accuracies at 45.80% each. These depths result in more splits, increasing the risk of overfitting to the training data while not improving generalization. A depth of 10 provides the best balance between simplicity and performance, since training time increases significantly with depth.

Classes 0 (airplane) and 8 (ship) generally show higher precision and recall across depths, which suggests that they might be easier to distinguish from other classes. Classes 5 (dog) and 7 (horse) exhibit moderate performance, while 2 (bird) and 3 (cat) have the highest rates of being misclassified across the board.

It is also observed that the custom implementation performs more evenly across all classes, whereas the Scikit-learn implementation has spikes in certain classes over others. This could be due to random variances in the data, and ends up averaging out to a ever so slight edge to the custom implementation. All other metrics such as precision, recall, and F1-measure indicate that both implementations perform similarly in this scenario.

3.3 Multi-Layer Perceptron (MLP)

The MLP classifier was evaluated in three configurations:

1. Shortened MLP which includes only the first layer
2. Base MLP with three layers (input, hidden, and output)
3. Extended MLP with additional layers added to the base configuration

The features were reduced to 50 dimensions via PCA in order to be passed through the input layer.

3.3.1 Shortened MLP

The shortened MLP had an accuracy of 69.20%.

Confusion Matrix:

```
[[62  2  1  2  2  1  1  0 26  3]
 [ 1 71  0  2  1  0  1  0  5 19]
 [10  0 50  4 11 10 13  1  1  0]
 [ 2  0  6 63  2 16  7  1  2  1]
 [ 3  0  4  7 59  4  4 17  1  1]
 [ 0  1  5 15  2 70  3  1  3  0]
 [ 1  0  6  8  3  1 80  0  0  1]
 [ 2  1  3  9  6  4  2 70  2  1]
 [ 7  1  0  0  0  0  1  0 85  6]
 [ 3  7  0  0  1  0  0  0  7 82]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.68	0.62	0.65	100
1	0.86	0.71	0.78	100
2	0.67	0.50	0.57	100
3	0.57	0.63	0.60	100
4	0.68	0.59	0.63	100
5	0.66	0.70	0.68	100
6	0.71	0.80	0.75	100
7	0.78	0.70	0.74	100
8	0.64	0.85	0.73	100
9	0.72	0.82	0.77	100
accuracy			0.69	1000
macro avg	0.70	0.69	0.69	1000
weighted avg	0.70	0.69	0.69	1000

3.3.2 Base MLP

The base MLP had an accuracy of 70.30%

Confusion Matrix:

```
[[70  2  4  0  1  0  1  0 18  4]
 [ 3 68  0  3  0  0  1  0  5 20]
 [ 8  0 55  5  9  9 12  1  1  0]
 [ 1  0  7 65  2 15  8  0  1  1]
 [ 1  0  5  8 60  3  4 15  2  2]
 [ 0  0  5 16  2 73  3  0  1  0]
 [ 1  0  6  6  4  2 80  0  1  0]
 [ 2  1  3  5 12  6  1 67  1  2]
 [12  1  3  0  0  0  0  0 77  7]
 [ 3  5  0  0  1  0  0  0  3 88]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.69	0.70	0.70	100
1	0.88	0.68	0.77	100
2	0.62	0.55	0.59	100
3	0.60	0.65	0.62	100
4	0.66	0.60	0.63	100
5	0.68	0.73	0.70	100
6	0.73	0.80	0.76	100
7	0.81	0.67	0.73	100
8	0.70	0.77	0.73	100
9	0.71	0.88	0.79	100
accuracy			0.70	1000
macro avg	0.71	0.70	0.70	1000
weighted avg	0.71	0.70	0.70	1000

3.3.3 Extended MLP

The extended MLP had an accuracy of 68.90%

```
Confusion Matrix:
[[78  0  4  0  0  0  1  0 12  5]
 [ 2 61  0  2  0  0  1  0  7 27]
 [10  0 44  3  6  9 23  5  0  0]
 [ 1  0  4 57  1 25 10  1  0  1]
 [ 4  0  1  7 54  3  8 21  2  0]
 [ 0  0  1 13  1 75  5  4  1  0]
 [ 3  0  3  5  1  1 86  0  1  0]
 [ 3  0  0  5  4 11  3 69  1  4]
 [14  0  0  1  2  0  0  0 76  7]
 [ 4  3  0  0  0  0  0  0  4 89]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.66	0.78	0.71	100
1	0.95	0.61	0.74	100
2	0.77	0.44	0.56	100
3	0.61	0.57	0.59	100
4	0.78	0.54	0.64	100
5	0.60	0.75	0.67	100
6	0.63	0.86	0.73	100
7	0.69	0.69	0.69	100
8	0.73	0.76	0.75	100
9	0.67	0.89	0.76	100
accuracy			0.69	1000
macro avg	0.71	0.69	0.68	1000
weighted avg	0.71	0.69	0.68	1000

3.3.4 Summary

The depth of the MLP has a direct influence on its ability to learn and generalize from data, and the base MLP achieves the highest accuracy at 70.30%. The shortened MLP, with fewer layers, lacks sufficient capacity to model complex features, resulting in slightly reduced accuracy at 69.20% and lower class-specific precision and recall. Conversely, the extended MLP scored slightly lower than the other MLPs at 68.90% accuracy. If we look at the architecture of the extended MLP, it is adding two additional hidden layers to the base MLP:

```
    additional_layers = [nn.Linear(10, 128), nn.ReLU(),
nn.Linear(128, 10)]
```

This should perform similarly or maybe slightly worse than the base MLP likely due to optimization challenges, it could also suggest overfitting taking place since its accuracy is not very far off the shortened MLP.

Varying the sizes of the hidden layers affects both computational cost and performance. Larger hidden layers allow the network to learn more intricate patterns, but will incur a higher computational cost and the risk of overfitting. For instance, the base MLP's balanced hidden layer sizes 50 → 512 → 512 → 10 achieve the best accuracy and generalization across all classes. Smaller hidden layers would reduce computational cost but introduces the risk of underfitting by failing to model complex data relationships.

The MLP models were trained using a fixed number of 20 epochs, a 0.01 learning rate, and a Cross-Entropy Loss function to measure classification performance [7]. An SGD optimizer with momentum 0.9 was also added to accelerate convergence while preventing oscillations in the gradient updates [8]. The models were trained on feature vectors with PCA-reduced dimensions to maintain critical features while ensuring efficient training. The training process leverages parallelism techniques by utilizing mini-batch gradient descent with a batch size of 128.

Each architecture was evaluated using accuracy, precision, recall, and F1-score to assess its performance. Upon investigating, the classification reports reveal that certain classes were more frequently misclassified across all MLP implementations. For example, class 2 (bird) with a precision of 62% and class 3 (cat) with a recall of 57% were challenging for all models. This can be attributed to the possibility of overlapping features or underrepresented patterns in the training set, as there are many varieties of both cats and birds and the sample set is not very extensive. Classes 6 (frog) and 9 (truck) on the other hand were well recognized, as they have a more distinct set of features despite variances.

In the context of facial image analysis, the MLP results show a consistently higher recall score which means it is correctly identifying positive instances, and minimizing false positives as indicated by the precision scores. This is by design as ReLU activation introduces non-linearity in the MLP, while Naive Bayes makes linear assumptions and Decision Trees make piecewise constant splits. The non-linearization in the MLP aids in learning the nuanced variations in human facial complexions.

3.4 Convolutional Neural Network (CNN)

3.4.1 Varying Convolutional Layers

A model was unable to be trained in time to evaluate how adding or removing convolutional layers influences the model's ability to learn and generalize from the data. However, from research, it is well-documented that the depth of a CNN directly influences its ability to capture hierarchical feature representations, with each additional layer contributing to the network's capacity to model increasingly abstract patterns. Simonyan and Zisserman [9] demonstrated in their seminal work on VGG architectures that deeper networks with smaller convolution filters (e.g. 3x3 kernels) significantly improve performance by enabling finer-grained feature extraction while maintaining computational efficiency. However, they also noted diminishing returns beyond a certain depth, as architectures that are too deep will inevitably lead to optimization challenges such as vanishing gradients.

He et al. [10] further addressed these limitations in their exploration of ResNets, introducing residual connections to mitigate vanishing gradients and improve training efficiency for very deep networks. It is emphasized from their findings that while adding layers can theoretically enhance learning, it must be balanced with mechanisms to ensure stable optimization and effective gradient flow. These insights suggest that while modifying the depth of VGG11 might improve accuracy, in the case of smaller datasets it is imperative to carefully tune and regularize the data to avoid overfitting or running into optimization challenges.

3.4.2 Kernel Size = 3

Below are the confusion matrix and classification report for the CNN model with a kernel size of 3 (which completed with 88.12% accuracy):

Confusion Matrix:

```
[[945   5   8   6   3   2   2   4  13  12]
 [ 10 936   0   0   0   2   0   1   5  46]
 [ 40   0 838  30  32  27  13  12   5   3]
 [ 24   1  43 738  24 112  19  23   5  11]
 [ 15   1  33  32 831  25  16  44   1   2]
 [  7   1  16  84  14 852   4  19   1   2]
 [  7   3  35  27  10  11 899   3   2   3]
 [  8   0   6  19  14  24   1 925   0   3]
 [ 53  12   1   6   0   3   1   0 909  15]
 [ 21  22   1   3   1   0   2   2   9 939]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.94	0.89	1000
1	0.95	0.94	0.94	1000
2	0.85	0.84	0.85	1000
3	0.78	0.74	0.76	1000
4	0.89	0.83	0.86	1000
5	0.81	0.85	0.83	1000
6	0.94	0.90	0.92	1000
7	0.90	0.93	0.91	1000
8	0.96	0.91	0.93	1000
9	0.91	0.94	0.92	1000
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

3.4.3 Kernel Size = 5

Below are the confusion matrix and classification report for the CNN model with a kernel size of 5 (which completed with 90.25% accuracy):

```
Confusion Matrix:
[[919  11  15  13   6   0  10   1  10  15]
 [  4 957   0   0   1   0   0   1   3  34]
 [ 28   1 833  34  37  30  27   3   4   3]
 [ 10   1  16 787  32 107  31   5   6   5]
 [  2   1  23  31 896  16  20  10   1   0]
 [  5   0   6  75  13 882   6  11   0   2]
 [  4   1  13  13   7   5 954   0   1   2]
 [  5   0   5  21  21  32   2 908   1   5]
 [ 22  13   3   5   2   2   2   0 927  24]
 [  7  16   1   3   2   0   3   1   5 962]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.92      0.92      1000
           1       0.96      0.96      0.96      1000
           2       0.91      0.83      0.87      1000
           3       0.80      0.79      0.79      1000
           4       0.88      0.90      0.89      1000
           5       0.82      0.88      0.85      1000
           6       0.90      0.95      0.93      1000
           7       0.97      0.91      0.94      1000
           8       0.97      0.93      0.95      1000
           9       0.91      0.96      0.94      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

Total training time: 20987.07 seconds
```

3.4.4 Summary

Using a smaller kernel size will capture more fine-grained details such as textures and edges, but struggles with capturing broader spatial patterns like with class 3 (cat) having slightly lower accuracy (78%) and recall (74%) when compared to other classes. A larger kernel should allow the model to capture broader spatial relationships, improving its ability to generalize [9]. Testing confirmed this, as an improvement in accuracy was observed by increasing the kernel size to 5 across all classes.

There does exist a computational cost trade-off when increasing the kernel size, as seen in the training time for kernel size 5 being significantly higher at 20,987 seconds versus 11,955 seconds for kernel size 3. Larger kernels incur a quadratic increase in computational operations for marginal performance gain.

Several optimization algorithms and techniques were used to enhance training efficiency and generalization. Like the MLP, the CNN implements a SGD optimizer with momentum of 0.9, and its own learning rate of 0.001, weight decay of 0.0005, and trains over 50 epochs. Momentum helps accelerate convergence and reduces oscillations in gradient updates as mentioned in the MLP summary section, while weight decay (L2 regularization) penalizes large weight values to prevent overfitting. The CrossEntropyLoss function is also utilized here to measure classification performance, as necessitated by the multi-class workload of CIFAR-10

classification. Furthermore, a ReduceLROnPlateau scheduler is incorporated to dynamically adjust the learning rate, lowering it when the program senses that performance has stagnated, helping to refine convergence in later epochs. Dropout(0.5) and batch normalization are implemented by VGG11 by default.

In the context of facial image analysis, these optimization techniques enhance the CNN's ability to extract meaningful features from the data. This optimization pipeline enables the CNN with a kernel size of 5 to achieve a test accuracy of 90.25%, far surpassing the base MLP accuracy of 70.30%, Naive Bayes' accuracy of 71.80%, and decision tree (with depth 10) accuracy of 48.20%. Unlike Naive Bayes and decision trees, which struggle with the hierarchical and spatial nature of image data, and MLPs which flatten input and lose spatial relationships, the CNN is able to model fine-grained details such as wrinkles in the skin and eye shapes, as well as broader spatial patterns like overall facial structure. This makes CNNs particularly effective for complex tasks like facial recognition.

4. References

[1] "CIFAR10 Dataset," PyTorch, [Online]. Available: https://pytorch.org/vision/main/generated/torchvision.datasets.CIFAR10.html. [Accessed: Nov. 17, 2024].

[2] "Data Loading and Processing Tutorial," PyTorch, [Online]. Available: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html. [Accessed: Nov. 17, 2024].

[3] "Torchvision Models," PyTorch, [Online]. Available: https://pytorch.org/vision/main/models.html. [Accessed: Nov. 17, 2024].

[4] "Torchvision Transforms," PyTorch, [Online]. Available: https://pytorch.org/vision/main/transforms.html. [Accessed: Nov. 17, 2024].

[5] "Naive Bayes with Scikit-Learn," DataCamp, [Online]. Available: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn. [Accessed: Nov. 17, 2024].

[6] "Gini Impurity and Entropy in Decision Tree ML," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/gini-impurity-and-entropy-in-decision-tree-ml/. [Accessed: Nov. 17, 2024].

[7] "torch.nn.CrossEntropyLoss," PyTorch Documentation, [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed: Nov. 17, 2024].

[8] "torch.optim.SGD," PyTorch Documentation, [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html. [Accessed: Nov. 17, 2024].

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv preprint arXiv:1409.1556, 2014. [Online]. Available: https://arxiv.org/pdf/1409.1556. [Accessed: Nov. 17, 2024].

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf. [Accessed: Nov. 17, 2024].