Exploring Various Convolutional Neural Network Architectures and Classifiers on CIFAR-10 Dataset

Cael Howard *UCI ID: 36813015*

David Joves *UCI ID: 37323591*

Diptanshu Sikdar *UCI ID: 61782612*

June 11, 2024

1 Summary

Our project aims to evaluate the performance of different classifiers on the CIFAR-10 image dataset. The tested models include K-nearest neighbors (KNN), logistic regression (LR), feed-forward neural networks (FFNN), and convolutional neural networks (CNN). For our CNNs, we implemented a vanilla CNN with 149,042 parameters and an advanced architecture, known as EfficientNetB2, with 9.2M parameters. Our experiments revealed that the best model was EfficientNetB2 with 78.1% test accuracy and 0.778 F1-score, and the second best model was the vanilla CNN which had 73.1% accuracy and 0.731 F1-score.

2 Dataset Description

The dataset that we used is CIFAR-10, a collection of 60,000 32x32-pixel color images evenly distributed among 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). For all models, we analyzed their classification performance by confusion matrices, accuracy and loss versus epoch plots, and accuracy and error rates. A research paper that uses this dataset, "Deep Residual Learning for Image Recognition" by He et al. introduces residual learning frameworks to ease the training of very deep networks, significantly improving accuracy in image recognition tasks.

3 Classifiers

This section contains descriptions for all model we implemented and tested on the CIFAR-10 dataset. Each subsection contains a high-level description, software used, and relevant parameters.

3.1 K-Nearest Neighbor

The K-Nearest Neighbor (kNN) classifier is a model that can predict the class label for any unseen data point. By computing the Euclidean distance of the data point to the other points around it, i.e., its neighbor(s), we can do majority voting to determine that data point's class label. Using libraries like Scikit-learn, NumPy, and Matplotlib, we created this model and augmented the dataset to achieve our results.The main parameter that we varied in this model was the number of neighbors K, whose value we varied between 1, 3, 5, 10, 50, and 110.

3.2 Logistic Regression

Logistic regression is a model that tunes parameter values in an attempt to minimize the loss across data in the training set. Typically, logistic regression is applied to binary classification problems because its logistic (sigmoid) activation function effectively differentiates between two distinct classes. However, applying this method to the CIFAR-10 dataset involves adjusting the model to include 10 logistic functions and applying a softmax function to ensure the sum of the likelihoods that a data entry is any particular class always sums to one.

3.3 Feed-forward Neural Network

A feed-forward neural network (FFNN) is a type of artificial neural network that consists of an input layer, one or more hidden layers, and an output layer (Figure 2). Each neuron in a layer is connected to every neuron in the subsequent layer through weighted connections. In the hidden and output layers, each neuron applies an activation function to the sum of inputs from the previous layer's neurons and respective weighted connections. FFNNs are used for various tasks, including classification and regression, by learning from data and adjusting weights through backpropagation to minimize error.

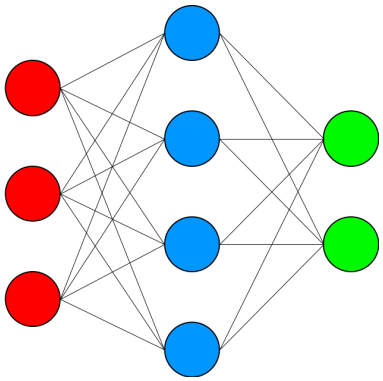

Figure 1: Feed-forward Neural Network (FFNN)

3.4 Convolutional Neural Network

The Convolutional Neural Network (CNN) is a special type of deep learning algorithm that has had exceptional success in several applications, including image classification and time-series prediction. Designed to perform salient feature extraction via convolutions and pooling operations, CNNs can learn both local and global spatial relationships within the data. CNNs are typically robust to overfitting and can be very efficient as they can take advantage of GPUs and multi-threading.

The internal architecture of a CNN usually consists of three layers: a convolutional layer, a pooling layer, and a fully connected layer. In the convolutional layer, the dot product of the filter and the input pixels is calculated, which is then repeated after the filter (filter is a set of weights that is used to scan for specific features of an image) shifts by strides until the entire image is scanned, as depicted in Figure 3.1. Each filter scans for a different pattern such as the diagonal line shown in the yellow square in Figure 3.2, and a unique feature map is formed for the whole image using that filter.

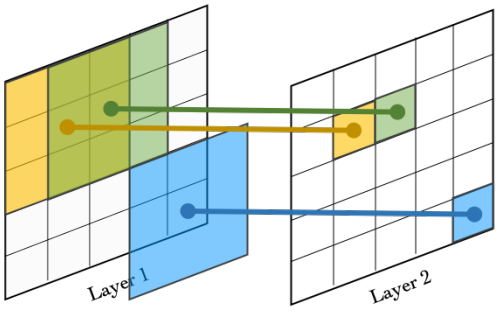

Figure 2: Sliding Window Approach

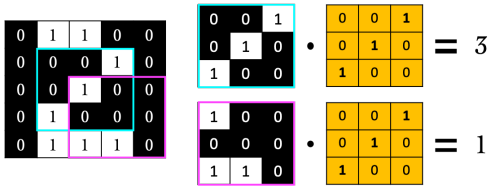

Figure 3: Convolution Operation on B/W Image

Additionally, the convolution filters lead to sparsity of connections in the neural network as each output value only depends on a small patch of inputs. For example, in an 2D image, a CNN may use a 3x3 convolution filter with a sliding window approach described above and depicted in Figure 1. Varying the width and height of the filters can alter the convolutional receptive field, shown in Figure 3.3. As the filter size increases, the more inputs pixels are mapped into the output, leading to more global context. Conversely, a smaller filter would identify more localized details.

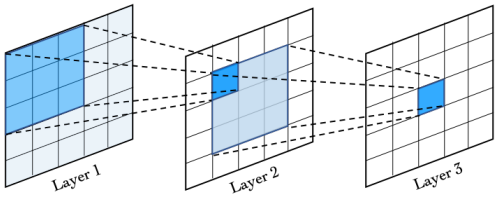

Figure 4: Receptive Field of a CNN

In deep CNNs, multiple convolutional layers can be stacked together to recognize more complex patterns. To reduce the dimensionality, complexity, and size of the model, pooling layers are used to compress information by scanning a filter across the image and using an aggregation function. Although pooling reduces the size of the feature map, it does not reduce the features themselves. There are various kinds of pooling methods, but the most commonly used methodology is Maximum Pooling returns the pixel with the maximum value to the output array. Finally, In the fully connected layer, an activation function is applied for classification of inputs based on the feature maps. Several fully connected layers can be used for classification of more advanced datasets.

The architecture of our CNN is detailed in Figure 5. It consists of 2 pairs of convolutional and max-pooling layers, followed by a fully-connected neural network with dropout to reduce overfitting and mapped to the 10 output classes.

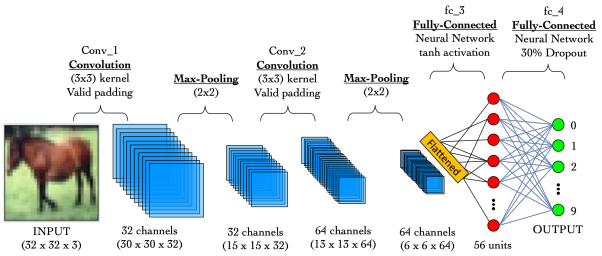

Figure 5: Standard CNN Architecture

4 Experimental Setup

This section contains each model's setup including it's experimental procedure, partitioning of data, and specific metrics we wanted to analyze.

4.1 Setup for kNN

For kNN, we focused on the classification accuracy and the model's error while varying K. First, we partitioned our data so that 60% of the CIFAR-10 dataset was used for testing, 20% for validation, and 20% for testing. After fitting the model, we produced predictions and computed accuracy scores for both the testing and training data sets. This was done by experimenting with the different K values we mentioned earlier.

4.2 Setup for Logistic Regression

To assess the quality of the logistic regression model, the key metrics recorded were training/validation accuracy, training/validation loss, and the relationship between predicted and real labels of the test dataset. The model was trained over 100 epochs, and we tracked its performance over time on a validation set in order to visualize the relationship between training and non-training data performance. To map the images into information that the model could process, we needed to "flatten" the data, meaning that the 32x32 matrix representing the image was converted to a 1024x1 array representing all of the pixels ordered sequentially. The values of the pixels were also normalized, being converted from values from a range of [0,225] to [0,1].

4.3 Setup for Feed-forward Neural Network

To assess the performance of the FFNN, the key metrics included training/validation accuracy and loss over 50 epochs as well as the relationship between predicted and real labels of the test dataset. We partitioned our data so that 60% of the CIFAR-10 dataset was used for testing, 20% for validation, and 20% for testing. To input the images into FFNN, we flattened and normalized the 32x32 image matrix into a 1024x1 array with values ranging from [0,1] instead of [0,255].

4.4 Setup for Standard Convolutional Neural Network

To evaluate the performance of the vanilla CNN, the key metrics included training/validation accuracy and loss over 50 epochs, and the relationship between predicted and real labels of the test dataset. We partitioned our data so that 60% of the CIFAR-10 dataset was used for testing, 20% for validation, and 20% for testing. After normalizing pixels to [0,1], we applied convolutional and pooling layers, before flattening the output and learning the neural network.

5 Experimental Results

This section contains the results of the respective models and includes it's metrics and interpretation. The accuracy and the F1 scores for all the models on the test dataset are summarized in the table below.

Model	Accuracy	F1 Score
KNN	0.336	0.331
LogReg	0.361	0.348
FFNN	0.465	0.458
CNN	0.731	0.731
EfficientNet	0.781	0.778

Table 1: Models Comparison

5.1 Results for kNN

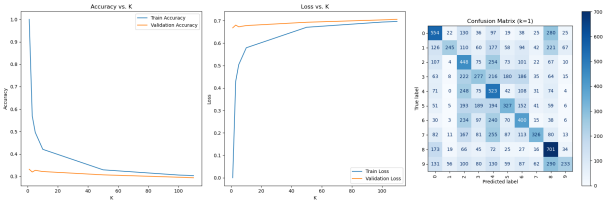

Figure 6: kNN Accuacy over K, Loss Over K, and Confusion Matrix

Figure 6 depicts the KNN classifier's performance. The left graph shows the accuracy over multiple k values, where both training and validation accuracy decrease as K increases. The middle graph illustrates the loss as K increases. The training loss increases significantly, indicating potential overfitting, while

validation loss remains steady. The highest accuracy occurs at K=1 and shows overfitting. The confusion matrix on the right shows classification results: class 0 (airplane) and class 8 (ship) have highest accuracy, while classes 3 (cat) and 4 (deer) are significantly misclassified, especially confused with class 7 (horse) and class 5 (dog).

5.2 Results for Logistic Regression

After training our logistic regression model, the test dataset accuracy was 36.14% with a loss of 1.91. This performance matched the model's behavior on the validation data after the last epoch, suggesting it accurately reflects performance on a random dataset. The training and validation errors are shown in Figure 5.2. One reason for overfitting could be the process of flattening the image before inputting it into the model. This process fails to properly account for horizontal and vertical shifts, causing the model to interpret images differently based on their position in the matrix. While more complex neural networks may identify these shifts, logistic regression is not complex enough to accurately interpret pixels as "features" to minimize loss.

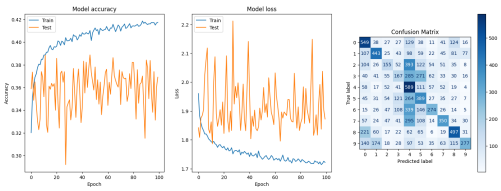

Figure 7: Logistic Regression Accuracy, Loss, and Confusion Matrix

5.3 Results for Feed-forward Neural Network

Figure 5.8 depicts the accuracy vs. epoch, loss vs. epoch, and confusion matrix for the FFNN's performance in the CIFAR-10 dataset. The left graph shows the model accuracy over 50 epochs, where the training accuracy gradually increases and stabilizes around 0.43, while the validation accuracy starts higher and slightly fluctuates, ultimately reaching a similar plateau at around 0.47. The middle graph depicts the model loss, with both training and validation losses decreasing over time, indicating improved learning. Since the validation loss did not cross over the training loss and started increasing, we can infer that the model has not overfitted. The confusion matrix on the right provides a detailed view of the classification results. The highest accuracy is observed for classifying class 7 (horse) with 593 correct predictions. Conversely, classes such as 3 (cat) and 5 (dog) show significant misclassifications across other classes. The overall performance suggests that while the model is learning and reducing errors, it struggles with certain classes.

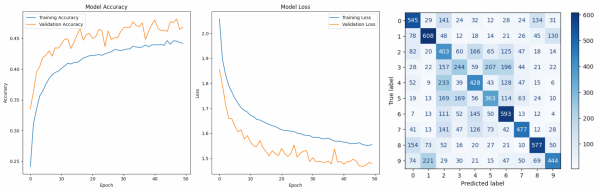

Figure 8: (FFNN) Accuracy vs Epoch, Loss vs Epoch, Confusion Matrix

5.4 Results for Regular CNN

Figure 9 depicts the accuracy vs. epoch, loss vs. epoch, and confusion matrix for the CNN' performance in the CIFAR-10 dataset. The left graph shows the model accuracy over 50 epochs, where the training accuracy rapidly increases and stabilizes around 0.9, while the validation accuracy fluctuates around 0.7. The middle graph illustrates the model loss, with the training loss decreasing significantly and stabilizing below 0.4, while the validation loss remains relatively steady around 0.9, indicating a potential overfitting issue. The confusion matrix on the right provides detailed classification results, with the highest accuracy observed for classifying class 1 (automobile) and class 9 (truck) with 834 and 830 correct predictions, respectively. In contrast, class 2 (bird) and class 3 (cat) show significant misclassifications. Overall, the CNN demonstrates strong training performance but requires further tuning to improve generalization and reduce overfitting on the CIFAR-10 dataset.

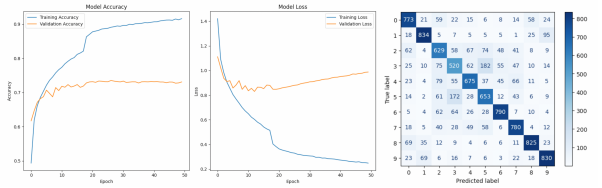

Figure 9: (CNN) Accuracy vs Epoch, Loss vs Epoch, Confusion Matrix

6 Insights

The results from the different classifiers aligned with the expected performance of each model. The worst-performing architectures were the K-nearest neigh-

bors and logistic regression models. Their reliance on discrete features that can be quantified demonstrated itself as a weakness when attempting to classify images from the CIFAR-10 dataset. On the other hand, the convolutional neural networks' flexibility in weighing and processing the pixels of the images allowed for a much higher accuracy on the test data. Both the manually trained and EfficientNet architectures scored significantly better than the other three models. An interesting behavior was the success that these weaker models had when identifying the ship class from the dataset, which was likely due to the consistent presence of blue pixels in the image. The opposite behavior was observed in the high inaccuracy of the bird class, which included images that had a wider variance of colors caused by different species and environments.

7 Contributions

This section highlights each member's contribution to this project. This includes what model's they worked on and how they contributted to the document.

7.1 Cael's Contribution

Cael worked on the Logistic Regression classifier. They have also contributed to it's respective sections in 3, 4 and 5, which includes plotting the graphs and computing the confusion matrices for their model. Additionally, they also worked on the Insights sections.

7.2 David's Contribution

David worked on the EfficientNetB2 and kNNclassifier. They have also contributed to it's respective sections in 3, 4 and 5 which includes plotting the graphs and computing the confusion matrices for their model.

7.3 Diptanshu's Contribution

Diptanshu worked on the Feed-forward Nueral Network and the Standard Convolutional Nueral Network. They also contributed to the document for the respective classifiers in sections 3, 4 and 5, including working on additional figures for all analyzed models.

8 Appendix

This section contains all information for our additional model, EfficientNetB2

8.1 EfficientNetB2

Another CNN model we looked at was EfficientNetB2. It leverages a combination of scaling dimensions—depth, width, and resolution—using a compound scaling method, making it both efficient and effective for image classification tasks. We modified the base model by excluding its top layers and adding custom layers. It's hyperparameters included an Adam optimizer with a learning rate of 0.0001, a batch size of 64, and data augmentation settings such as rotation range (15 degrees), width and height shift range (0.1), horizontal flip, and zoom range (0.1).

8.2 Setup for EfficientNetB2

With the key focus of recording data such as the training/validation accuracy and losses, we aim to assess the model in it's performance of classifying CIFAR-10. We modified the base model, provided by the TensorFlow framework, by excluding its top layers and adding custom layers: a global average pooling layer, a dense layer with 256 units and ReLU activation, a dropout layer with a rate of 0.5, and a final dense layer with 10 units and softmax activation. The model was trained for 50 epochs with 60% of the data allocated for training, 20% for validation, and 20 for testing, ensuring a comprehensive evaluation of performance.

8.3 Results for EfficientNetB2

Figure 5.5 depicts the accuracy vs. epoch, loss vs. epoch, and confusion matrix for the EfficientNetB2's performance on the CIFAR-1 0 dataset. The left graph shows the model accuracy over 50 epochs, where the training accuracy rapidly increases and stabilizes around 0.85, while the validation accuracy fluctuates significantly around 0.75. The middle graph illustrates the model loss, with the training loss decreasing significantly and stabilizing below 0.5, while the validation loss shows considerable fluctuation, indicating a potential overfitting issue.

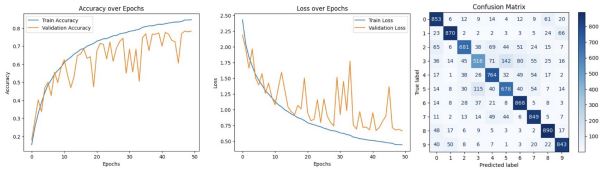

Figure 10: (EffcientNetV2) Accuracy vs Epoch, Loss vs Epoch, Confusion Matrix

The confusion matrix on the right provides detailed classification results, with the highest accuracy observed for classifying class 1 (automobile) and class 6 (frog) with 870 and 868 correct predictions, respectively. In contrast, class 3 (cat) and class 9 (truck) show significant misclassifications, particularly confusing with class 5 (dog) and class 1 (automobile), respectively. Overall, EfficientNetB2 demonstrates strong training performance but requires further tuning and regularization techniques to improve generalization and reduce overfitting on the CIFAR-10 dataset.

References

Fabris, A. (n.d.). *Efficientnet v2 image classification - 93% accuracy* [Retrieved June 10, 2024]. https://www.kaggle.com/code/annafabris/efficientnet-v2-image-classification-93-accuracy

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Keras. (n.d.). *Efficientnetv2* [Retrieved June 10, 2024]. https://keras.io/api/keras_cv/models/backbones/efficientnetv2/

Papers with Code. (n.d.). *Efficientnetv2* [Retrieved June 10, 2024]. https://paperswithcode.com/method/efficientnetv2