Kolby Boyd

ITAI 1378

Prof. McManus

L07

Object Detection Lab

Pre Lab Setup:

I downloaded the notebook from the course page on Canvas and chose to run it in Google Colab for convenience. Then, I installed all of the necessary libraries and modules. These include: TensorFlow, NumPy, Scikit-Learn, and MatPlotLib.

Luckily, I did not need to install additional libraries or modules as they were all included in the Notebook.

Model Implementation:

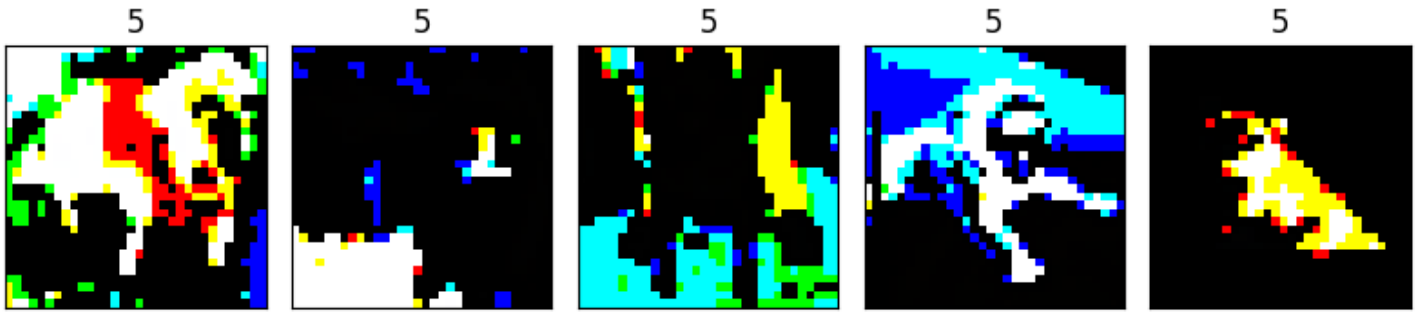
I loaded the CIFAR-10 dataset using TensorFlow's Keras API. After loading, I encountered an error in the prewritten code. I needed to load and unpack the CIFAR-10 dataset using the 'load_data()' function from the 'tf.keras.datasets.cifar10' module. This should return four values: 'x_train', 'y_train', 'x_test', and 'y_test'. My updated code is: '(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()'.

After this step, we needed to flatten the labels. This means reshaping the label arrays from a 3D shape into a 2D shape. In the CIFAR-10 dataset, the labels need to be flattened for convenience and compatibility with the machine learning algorithms and loss functions. A 2D array is much easier to work with than a 3D array and Scikit-learn's API is designed to work seamlessly with 2D arrays for labels.

Next, preprocessing of the data. Scaling the data to have a mean of 0 and a variance of 1 (we do this with the 'StandardScaler' in Scikit-learn), is a common preprocessing step in machine learning. This process is also known as z-score normalization or standardization. Standardization centers the data around zero, which can help to remove any bias and ensure that each feature has a similar scale. This process also increases the model performance for algorithms that are sensitive to the scale of the features (such as k-NN or k-nearest neighbors).

Finally, using the 'display_images' function, we learned to display images along with their labels, which is helpful for visual verification of the data. This function is helpful for visually inspecting the images in the dataset, which allows us to verify that the images and labels are loading correctly and processing true, before proceeding with further analysis.

The CIFAR-10 dataset is made up of 60k 32x32 color images within 10 different classes, each class having 6k images. 1=airplane, 2=automobile, etc. I chose "dog" which is category # 5. The next code block displays 5 images from the "dog" category using the display_images function.



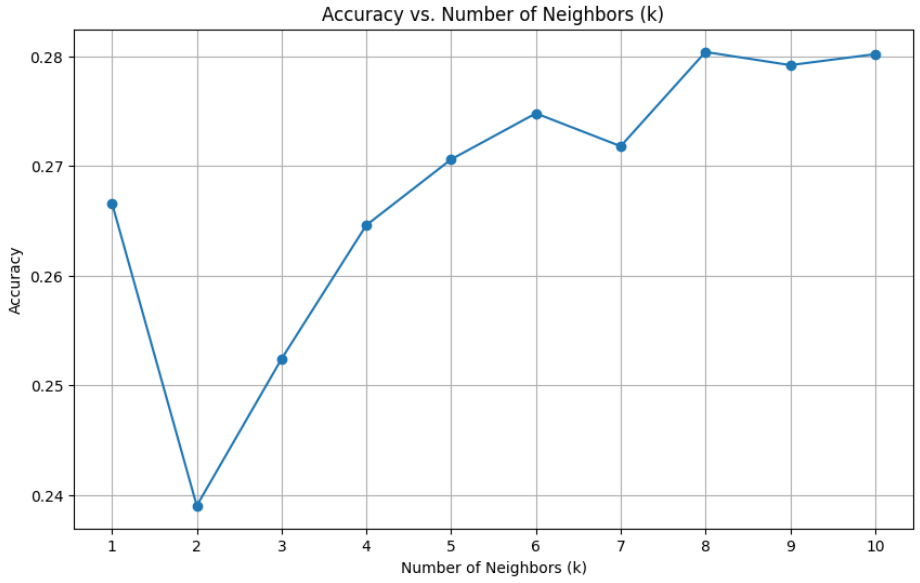
Next, training the k-NN classifiers for different values of k. First, we imported ‘KNeighborsClassifier’ from scikit-learn to create a k-NN classifier. Then we import ‘cross_val_score’ to perform the cross validation. We have defined a list of k values (number of neighbors) for which we want to evaluate that classifier. As we loop over each value of k, we create a k-NN classifier with that same value of k and evaluate its performance using 5-fold cross validation. Finally, we calculate the mean accuracy across all folds for each value of k and store it in the ‘cv_scores’ list.

This step took quite a while, almost a full hour on my system. I could have chosen specific k values in the range from 1-10 (2, 4, 6, 8, 10) to get closer to the best k value, instead of running all values from 1-10 in order to speed up this process. However, I decided to look up a way to implement parallel processing to accomplish this. I wanted to use all available CPU cores for faster computation, so I set the ‘n_jobs’ = to -1 in both ‘KNeighborsClassifier’ and ‘cross_val_score’. Then I added a block to print the different values of k stored in ‘cv_scores’.

Once finally completed, I used a new code snippet to determine the best value of k.

```
for k, score in zip(k_values, cv_scores):
    print(f"K = {k}: Accuracy = {score}")
```

In this, case the best value of k is 8, as shown by the graph below:



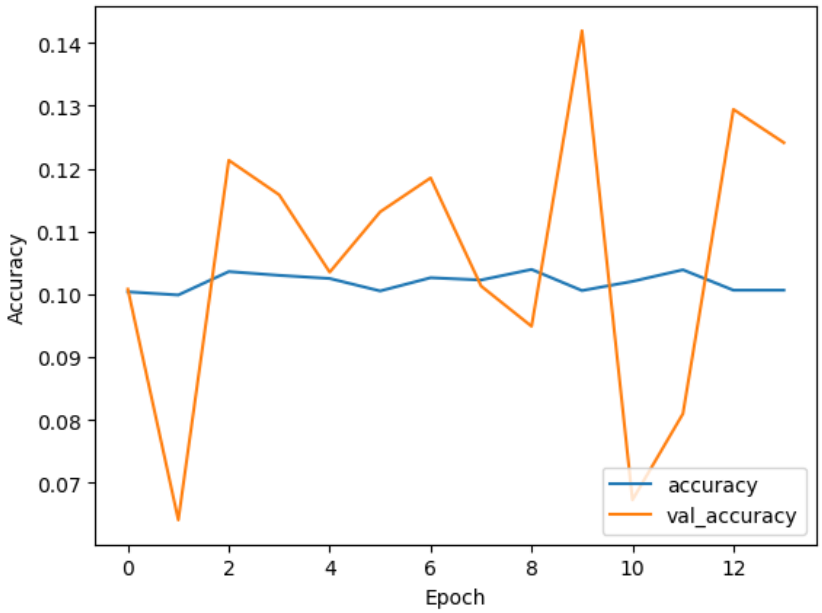
Let's look into how bias and variance come into play. First, we look at the calculation of the misclassification errors. With the different values of k, there is a bias-variance trade-off. A higher misclassification error indicates a higher bias, called underfitting, while a low misclassification error, suggests overfitting, or a higher variance. The goal is to find an optimal k value that balances both bias and variance to achieve the best general performance on unseen data.

Now, we will fine tune the model using GridSearchCV. This technique is used so we can perform the grid search over different values of k to find the optimal k value that will minimize the effects of overfitting and underfitting. GridSearchCV is trying out different values of k and will eventually select the one that maximizes the model’s performance, based on accuracy in this case. GridSearchCV is using cross validation to ensure that the model’s performance is evaluated sufficiently well across multiple parts of the training data.

After applying GridSearchCV, the most optimal ‘k’ value is 8, with an accuracy of 27.73%. This low accuracy score can be improved in a few different ways. Tuning the hyperparameters, normalizing the data and preprocessing further could be of use in this application.

Model Evaluation:

The next code blocks begin to perform several tasks as part of model evaluation. The CIFAR-10 dataset is loaded in and the pixel values are normalized to a range between 0 and 1, which helps to speed up the training and improves the convergence of the model. Data augmentation using techniques like rotation, horizontal flipping and shifting are next. These increase the diversity of the training data by generating new images with variations, which help the model to learn more about features and improve its ability to generalize. Using TensorFlow/Keras, a convolutional neural network model is defined using the Sequential API. The model has multiple convolutional layers followed by batch normalization, max-pooling, dropout and fully connected layers. This is commonly used architecture in image classification. Then we compile the model by specifying the optimizer (Adam), the loss function (sparse_categorical_crossentropy for multi-class classification), and the evaluation metric (accuracy). This prepares the model for training.



During training, the model stopped after 14 epochs. This is likely due to early stopping triggered by the ‘EarlyStopping’ callback when the validation loss did not improve for the specified number of epochs (5 in this case). In the above graph, we can see that the validation loss and validation accuracy show fluctuations over epochs, with no clear trend toward improvement or degradation.

Summary:

Overall, the lab provided hands-on experience with implementing, fine-tuning, and evaluating machine learning models for object detection tasks, with a solid focus on understanding concepts such as preprocessing, model architecture and bias-variance trade-off.