Project bootcamp

Project idea: using feature engineering to explain results in MNIST and CIFAR-10(explainable AI).

**MNIST:**

Here I will show you the model of feature engineering on mnist dataset using svm.



**Description:**

**Imports:**

1. **TensorFlow and Keras:** These libraries are used for building and training the baseline Convolutional Neural Network (CNN) model.
2. **mnist:** This provides access to the MNIST dataset of handwritten digits.
3. **skimage.feature.hog:** This function computes the Histogram of Oriented Gradients (HOG) features from images.
4. **sklearn.svm.SVC:** This implements the Support Vector Machine (SVM) classifier, which we will use with the engineered HOG features.
5. **sklearn.metrics.accuracy\_score:** This function calculates the accuracy of the model's predictions.
6. **matplotlib.pyplot:** Used for plotting and visualizing the HOG features.
7. **numpy:** A fundamental library for numerical computations in Python.

**Load and Preprocess MNIST Data:**

* **Load MNIST:** The mnist.load\_data() function loads the MNIST dataset, splitting it into training and testing sets.
* **Reshape and Normalize:**
  + reshape: The image data is reshaped to include a channel dimension (1 for grayscale images) to match the CNN's input expectations.
  + astype('float32') / 255: The pixel values (originally 0-255) are converted to floats and normalized to the range 0-1.
* **One-hot Encode Labels:**
  + to\_categorical: The labels (representing digit classes 0-9) are converted into one-hot encoded vectors, where each label is represented as a 10-dimensional vector with a 1 at the index corresponding to the digit class.

**Baseline CNN Model:**

* **Sequential Model:** A Sequential model is created to stack layers linearly.
* **Flatten Layer:** The Flatten layer converts the 2D image data into a 1D vector, preparing it for the dense layers.
* **Dense Layers:**
  + The first Dense layer has 128 neurons and uses the ReLU activation function.
  + The second Dense layer has 10 neurons (one for each digit class) and uses the softmax activation function to produce probability distributions over the classes.
* **Compilation:**
  + loss='categorical\_crossentropy': Appropriate for multi-class classification.
  + optimizer='adam': A popular optimization algorithm for updating model weights during training.
  + metrics=['accuracy']: Tracks the classification accuracy during training and evaluation.
* **Training:** The fit method trains the model on the training data for 5 epochs (iterations over the entire dataset) with a batch size of 128. It also evaluates the model on the validation data (X\_test, y\_test) after each epoch.

**Evaluate Baseline Model:**

* The evaluate method assesses the model's performance on the test data, returning the loss and accuracy.
* The baseline accuracy is printed.

**Feature Engineering: HOG Features:**

* **HOG Feature Extraction:**
  + The hog function from scikit-image is used to extract HOG features from each image in the training and testing sets.
  + orientations=9: The number of orientation bins for the histograms.
  + pixels\_per\_cell=(8, 8): The size of each cell in the image grid.
  + cells\_per\_block=(2, 2): The number of cells in each block used for normalization.

**Train SVM on HOG Features:**

* An SVM classifier is instantiated.
* The fit method trains the SVM on the HOG features (train\_hog\_features) and the corresponding labels (y\_train.argmax(axis=1) extracts the class with the highest probability from the one-hot encoded labels).

**Predict and Evaluate:**

* The trained SVM is used to predict labels for the test HOG features.
* The accuracy\_score function calculates the accuracy by comparing the predicted labels (y\_pred) with the true labels (y\_test.argmax(axis=1)).
* The accuracy using HOG features is printed.

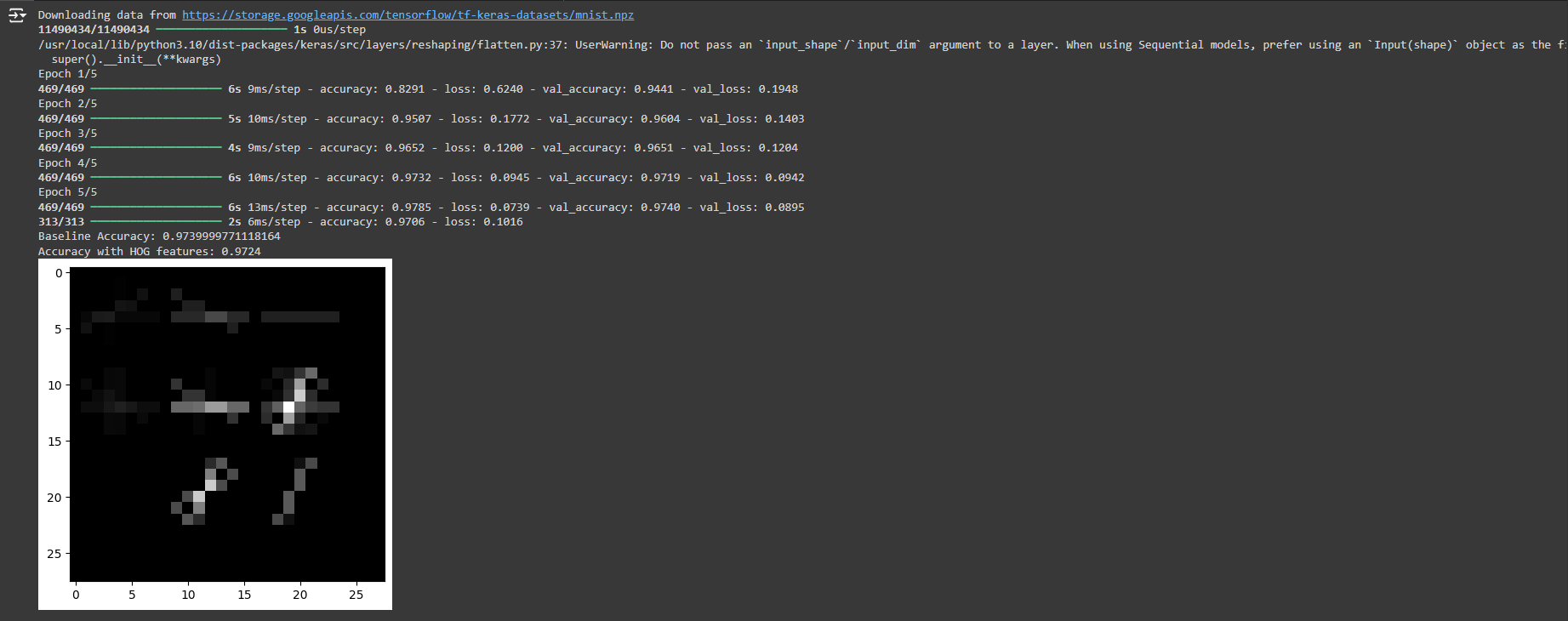
**Visualize HOG Features:**

 A sample image from the test set is selected.

 HOG features are extracted from this sample image, and the visualization is also generated using visualize=True.

 The HOG visualization is displayed using plt.imshow.

**OUTPUT:**

****

 The baseline CNN model achieved a high accuracy of about 97.4% on the MNIST dataset.

 The SVM model trained on HOG features also achieved a comparable accuracy of about 97.2%.

 The visualization of HOG features shows that it captures the edges and orientations within the image, which are important for recognizing handwritten digits.

**Feature Engineering as Explainable AI:**

* HOG features provide a more interpretable representation of the image data compared to the raw pixel values.
* By training a simpler model (SVM) on HOG features and achieving good accuracy, we gain some insight into what features are important for the model's decision-making.
* The HOG visualization further helps us understand how the model "sees" the image and what patterns it's focusing on.

**CIFAR-10:**

We apply feature engineering using edge detection on CIFAR-10.



**Load and Preprocess CIFAR-10 Data**

* **Load CIFAR-10:** The cifar10.load\_data() function loads the CIFAR-10 dataset, separating it into training and testing sets (X\_train, y\_train and X\_test, y\_test, respectively).
* **Preprocess Data:**
  + astype('float32') / 255: The pixel values (originally 0-255) are converted to floats and normalized to the range 0-1, which is a common practice in image processing for neural networks.
  + to\_categorical: The labels (representing image classes 0-9) are converted into one-hot encoded vectors. This transformation is necessary for multi-class classification problems, where each label is represented as a 10-dimensional vector with a '1' at the index corresponding to the class and '0' elsewhere.

**Baseline CNN Model**

* **Sequential Model:** A Sequential model is created, which is a linear stack of layers.
* **Convolutional Layers (Conv2D):**
  + The first Conv2D layer has 32 filters, each with a 3x3 kernel size. It uses the ReLU activation function and expects input images of shape (32, 32, 3) - 32x32 pixels with 3 color channels (RGB).
  + Convolutional layers are fundamental in image processing as they learn to detect spatial patterns and features in the images.
* **MaxPooling Layer (MaxPooling2D):**
  + A MaxPooling2D layer with a pool size of 2x2 is added.
  + MaxPooling reduces dimensionality by downsampling the feature maps, helping to make the network more robust to small shifts in the image.
* **Flatten Layer:**
  + The Flatten layer converts the 3D output from the convolutional layers into a 1D vector, preparing it for the fully connected (dense) layers.
* **Dense Layers:**
  + The first Dense layer has 64 neurons and uses the ReLU activation function.
  + The second Dense layer has 10 neurons (one for each class) and uses the softmax activation function to produce probability distributions over the classes.
* **Compilation:**
  + loss='categorical\_crossentropy': This is the loss function used for multi-class classification problems.
  + optimizer='adam': The Adam optimizer is employed to update the model's weights during training.
  + metrics=['accuracy']: The accuracy metric is used to track the model's performance during training and evaluation.
* **Training:**
  + The fit method trains the model on the training data for 5 epochs with a batch size of 128.
  + The validation\_data argument allows the model to be evaluated on the test set after each epoch to monitor its performance on unseen data.

**Evaluate Baseline Model**

* The evaluate method is used to assess the model's performance on the test set, returning the loss and accuracy.
* The baseline accuracy is printed.

**Feature Engineering: Edge Detection**

* **Sobel Filter:** The filters.sobel function from scikit-image is applied to each image in the training and testing sets to perform edge detection.
* **Edge Detection:** This process highlights the boundaries and outlines of objects within the images, potentially making them easier for the model to recognize.

**Train Model on Edge Features**

* **New CNN Model:** A new Sequential model is created, similar to the baseline model but with the input\_shape adjusted to (32, 32, 1) to accommodate the grayscale edge-detected images.
* **Reshape Edge Features:** The np.expand\_dims function adds a channel dimension to the edge features, making them compatible with the CNN's input expectations.
* **Training:** The new model is trained on the edge-detected images (train\_edge\_features) and their corresponding labels (y\_train) for 5 epochs with a batch size of 128. The test set (test\_edge\_features, y\_test) is used for validation.

**Visualize Edge Features**

* **Sample Image:** An image is selected from the test set.
* **Edge Detection:** The Sobel filter is applied to the sample image to generate its edge representation.
* **Visualization:** The edge-detected image is displayed using plt.imshow.

**In summary**, this code demonstrates the use of feature engineering (edge detection) to potentially improve the performance and interpretability of a CNN model on the CIFAR-10 image classification task. By highlighting object boundaries, the edge-detected images may provide the model with more salient information for distinguishing between different classes. The visualization of the edge features allows us to gain insights into what the model is focusing on when making its predictions.

**OUTPUT:**

This output shows the training process and evaluation results of a Convolutional Neural Network (CNN) model on the CIFAR-10 dataset, along with a comparison to a baseline model's accuracy on MNIST.

* **Training Progress:**
  + Each Epoch line represents one complete pass through the training data.
  + 391/391: Indicates the number of batches processed in each epoch (391 batches in total).
  + 39s 99ms/step: Shows the average time taken to process one batch.
  + accuracy: The accuracy of the model on the training data for that epoch.
  + loss: The loss value (categorical cross-entropy) on the training data.
  + val\_accuracy: The accuracy of the model on the validation data (a portion of the training data not used for training, used to assess how well the model generalizes).
  + val\_loss: The loss value on the validation data.
* **Evaluation Results:**
  + 313/313: Indicates the number of batches processed during the final evaluation on the test set.
  + 3s 8ms/step: The average time taken to process one batch during evaluation.
  + accuracy: 0.6001 loss: 1.1431: The final accuracy and loss of the model on the test set.
  + Baseline Accuracy: 0.5961999893188477: This is the accuracy of a different model (presumably a baseline model) on the MNIST dataset.

**Why MNIST Accuracy is Better**

Generally, MNIST tends to have higher accuracies compared to CIFAR-10 due to the following reasons:

1. **Task Complexity:**
   * MNIST is a simpler task involving classifying handwritten digits (0-9), which have relatively distinct shapes and fewer variations.
   * CIFAR-10 involves classifying objects from 10 different classes (e.g., airplanes, cars, birds, dogs), which have more complex shapes, variations in pose, and can be occluded in the images, making the task more challenging.
2. **Image Size and Color:**
   * MNIST images are grayscale and smaller (28x28 pixels), requiring less computational resources and potentially leading to faster training and easier feature extraction.
   * CIFAR-10 images are color and larger (32x32 pixels), increasing the complexity and requiring more sophisticated models to capture the relevant features.
3. **Model Architecture:**
   * The baseline model mentioned here might be a relatively simple CNN, which could perform well on MNIST due to its simplicity but might struggle to capture the intricacies of CIFAR-10.
   * More complex models with deeper architectures and techniques like data augmentation are typically needed to achieve high accuracy on CIFAR-10.

**In summary:** The output you provided shows the training and evaluation of a CNN model on CIFAR-10. The lower accuracy compared to the MNIST baseline accuracy is expected due to the inherent differences in task complexity, image characteristics, and potentially the model architecture used.