**Background and Method Introduction:**

**Overview of Linear Classifier:**

Linear classifiers are a fundamental concept in machine learning, used for classifying data points into different categories based on a linear decision boundary. This boundary, typically a straight line in 2D space (higher dimensions for more features), separates the data points based on their characteristics. The classifier takes data points as input, which are represented by their features.

In image classification, features might be numerical values derived from pixel intensities, textures, or other image properties. Each feature corresponds to a dimension in the feature space. For example, a colour image with R, G, and B channels would have a 3D feature space. The classifier has a weight vector, where each weight corresponds to a feature's importance in differentiating classes. Higher weights reflect greater influence on the decision boundary. A bias term is also used to adjust the position of the decision boundary, shifting it along the feature space.

The classifier performs a linear combination of the features, essentially multiplying each feature value by its respective weight and adding them together with the bias. This linear score doesn't directly provide a class prediction. The classifier uses an activation function to transform it into a class probability or label. Common choices include a sigmoid function which outputs a value between 0 and 1, representing the probability of belonging to a particular class. Based on the output of the activation function (e.g., highest probability class), the data point is assigned to a specific class.

**Applications in Image Classification:**

Binary Image Classification: Linear classifiers can be used for binary image classification tasks where the goal is to classify images into two categories, such as cat vs. dog classification or benign vs. malignant tumour detection in medical imaging.

Feature Extraction: Linear classifiers can be employed as feature extractors in more complex models. By using linear classifiers, it's possible to extract discriminative features from images which can then be fed into deeper neural networks or other classifiers for improved performance.

Multi-class Classification: While linear classifiers are inherently binary classifiers, they can be extended to handle multi-class classification tasks using techniques like one-vs-rest or one-vs-one classification. In image classification, this can be applied to classify images into multiple categories, such as recognizing different types of fruits in an image.

Dimensionality Reduction: Linear classifiers can also be used for dimensionality reduction in image classification tasks. Techniques like Linear Discriminant Analysis (LDA) or Principal Component Analysis (PCA) can be employed to reduce the dimensionality of image features while preserving the most discriminative information.

**Dataset and Tasks Description:**

**About CIFAR-10 Dataset:**

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks in machine learning. It consists of 60,000 colour images, evenly distributed across 10 different object classes i.e., aeroplane, automobile (excluding trucks), bird, cat, deer, dog, frog, horse, ship, truck. Each image is 32x32 pixels in size and represented in RGB format. The dataset is further split into 50,000 training images and 10,000 test images. Additionally, there are 10,000 labelled images from the same distribution used for validation.

**Tasks:**

1. Data Loading:

* The CIFAR-10 dataset is loaded using Keras' built-in dataset module.
* The dataset is split into training and testing sets ((X\_train, y\_train), (X\_test, y\_test)).

2. Data Preprocessing:

* Reshaped the input data (X\_train and X\_test) to flatten it into a 1D array and then normalised it by dividing by 255.0.
* Converted the target labels (y\_train and y\_test) into one-hot encoded format using np.eye(10).
* A validation set (X\_val, y\_val) is created by taking a portion (10%) of the training data.

3. Linear Classifier Class:

* A class named LinearClassifier is defined.
* It has methods for initialization (\_\_init\_\_), forward pass computation (forward), backward pass computation (backward), training (train), and prediction (predict).

4. Training the Linear Classifier:

* An instance of the LinearClassifier class is created.
* The train method of the LinearClassifier class is called to train the model using the training data (X\_train, y\_train) and validate it using the validation data (X\_val, y\_val).
* The training process involves iterating through epochs and mini-batches, computing forward and backward passes, updating weights and biases using gradient descent, and monitoring validation accuracy.

After training, the trained model is evaluated on the test set (X\_test, y\_test) to measure its performance using accuracy.

5. Improved Linear Classifier Class:

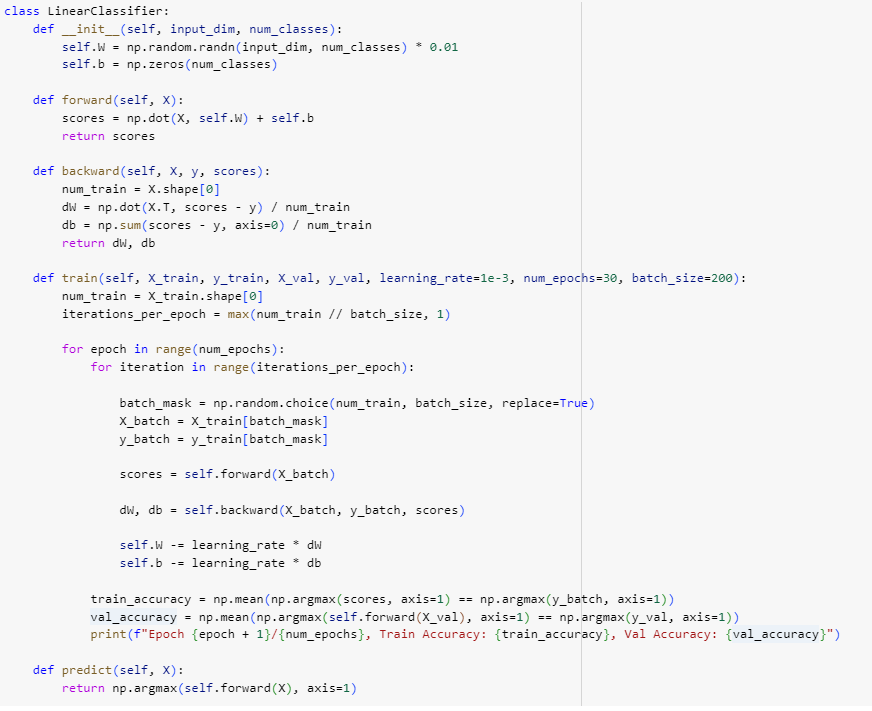
* A class named LinearClassifierImproved is defined, which is an improved version of the previous linear classifier.
* It adds regularisation to the backward pass computation to prevent overfitting.

6. Hyperparameter Tuning:

* Multiple hyperparameters such as learning rates, regularisation strengths, number of epochs, and batch sizes are experimented with to find the best combination that maximises validation accuracy.
* The model with the highest validation accuracy is selected as the best model.
* Finally, the test accuracy of the best model is calculated on the test set to evaluate its performance on unseen data.

**Algorithms Used:**

**LinearClassifier:**

****

1. Initialization (\_\_init\_\_):

* In this method, self.W represents the weight matrix of shape (input\_dim, num\_classes), where input\_dim is the dimensionality of the input features and num\_classes is the number of output classes.
* self.b represents the bias vector of shape (num\_classes,).
* We initialise self.W with small random values to break symmetry and prevent the network from getting stuck. Multiplying by 0.01 scales the random values to be small.
* We initialise self.b to zeros since we do not want any bias initially.

1. Forward Pass (forward):

* This method computes the raw scores for each class for a given set of input features X.
* It performs a dot product between the input features X and the weight matrix W, and then adds the bias vector b to obtain the raw scores.
* The output is a matrix of shape (num\_samples, num\_classes), where each row contains the raw scores for each class for a particular sample.

1. Backward Pass (backward):

* This method computes the gradients of the loss with respect to the weights W and bias b.
* It takes the input features X, the true labels y, and the scores computed in the forward pass as input.
* It calculates the gradients using the formula for the gradient of the softmax loss function with respect to the weights and bias.

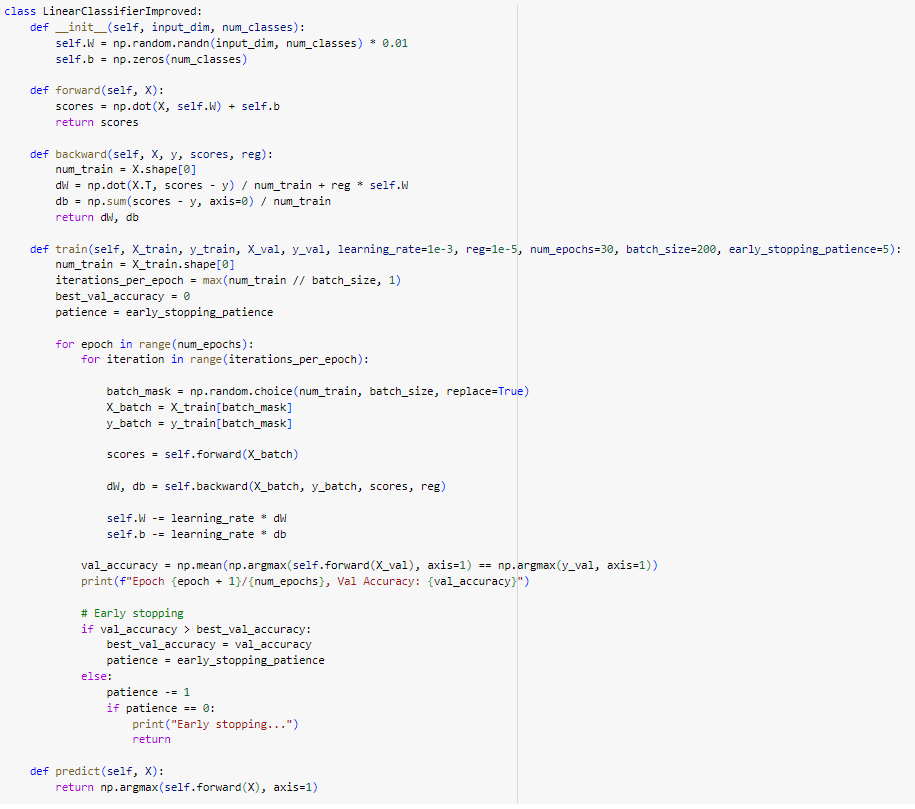
1. Training (train):

* This method trains the linear classifier using stochastic gradient descent (SGD).
* It iterates over the training data for a specified number of epochs and updates the weights and bias using gradients computed in the backward pass.
* It also prints the training and validation accuracies for each epoch to monitor the training progress.

1. Prediction (predict):

* This method predicts the class labels for a given set of input features X.
* It selects the class with the highest score computed in the forward pass for each sample.

**LinearClassifierImproved:**

****

1. Initialization (\_\_init\_\_):

* Similar to LinearClassifier, this method initialises the weights W and bias b.

1. Forward Pass (forward), Backward Pass (backward), and Prediction (predict):

* These methods are similar to their counterparts in LinearClassifier.

1. Training (train):

* In addition to the functionality in LinearClassifier, this method includes L2 regularisation and early stopping.
* L2 regularisation is implemented in the backward pass by adding a regularisation term to the gradients of the weights.
* Early stopping is implemented by monitoring the validation accuracy. Training stops if the validation accuracy does not improve for a certain number of epochs.

**Classification Results:**

|  | Linear Classifier | Improved Linear Classifier |
| --- | --- | --- |
| Best Validation Accuracy | 38.04 | 0.398 |
| Test Accuracy | 37.77 | 0.3878 |

Normal linear classifiers were able to achieve an accuracy of 37.77% on test data, while our improved model was able to achieve a performance of 38.78%. And the improved linear classifier also reduces overfitting, as we have used L2 regularisation which is a hyperparameter and tuned for better performance.

**Methods of Improvement:**

1. L2 Regularization:

LinearClassifierImproved includes L2 regularisation in the backward pass to prevent overfitting. This regularisation term penalises large weights and encourages the model to learn simpler patterns, leading to better generalisation performance.

1. Early Stopping:

LinearClassifierImproved implements early stopping during training to prevent overfitting and improve efficiency. If the validation accuracy does not improve for a certain number of epochs (specified by early\_stopping\_patience), training is stopped early. This helps in avoiding unnecessary computations and prevents the model from overfitting to the training data.

1. Learning Rate and Regularization Strength:

In LinearClassifierImproved, the learning rate (learning\_rate) and regularisation strength (reg) are passed as hyperparameters to the train method. This allows for more flexibility in tuning these parameters and optimising the model's performance.

By exploring different combinations of learning rates and regularisation strengths just like in grid search, LinearClassifierImproved can find the best parameters and achieve better performance.

1. Batch Size Adjustment:

LinearClassifierImproved supports training with mini-batch gradient descent by specifying the batch size (batch\_size) as a hyperparameter. Training with mini-batches can accelerate convergence and improve efficiency, especially for large datasets.

By testing different batch sizes during grid search, LinearClassifierImproved can find the optimal batch size that balances convergence speed and memory usage.