DL HW2

312511052 李元中

Training Flow:

Loading the Dataset:

- The code supports two datasets: CIFAR-10 and MNIST.
- The `database` variable determines the dataset choice. In this case, CIFAR-10 is chosen.
- The data is loaded using `cifar10.load_data()` and preprocessed.

Data Preprocessing:

- The image data is normalized to a range of [0, 1].
- For CIFAR-10, one-hot encoding is applied to the target labels (`y_train` and `y_test`).
- Data augmentation is performed on the training set using `ImageDataGenerator` to create variations of the images.

Model Architecture:

- The model is a sequential model with the following layers:
- Convolutional layer with 32 filters, kernel size (5,5), ReLU activation, and batch normalization.
- Another similar convolutional layer with batch normalization.
- Max pooling and dropout layers to reduce spatial dimensions and prevent overfitting.
- The same pattern is repeated with increased filter size for subsequent convolutional layers.
- Flatten layer to convert the 3D feature maps to 1D.
- Fully connected dense layer with ReLU activation and dropout for regularization.
- Output layer with softmax activation for multiclass classification.

Model Compilation:

- The model is compiled using the Adam optimizer and categorical cross-entropy loss, considering it is a classification task.
- The accuracy is also monitored during training.

Training the Model:

- The model is trained using the `fit` method.
- Data is provided through data generators for training and validation sets.
- A custom callback (`myCallback`) is defined to stop training when the accuracy reaches 99.5% and to evaluate the model on the test set after each epoch.

Model Evaluation:

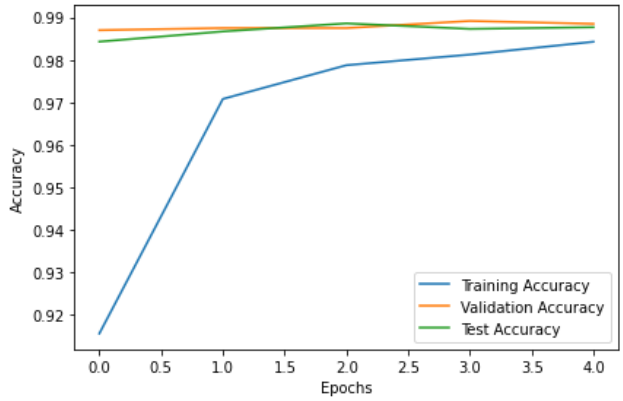
- After training, the model is evaluated on the test set using the `evaluate` method, and test loss and accuracy are printed.

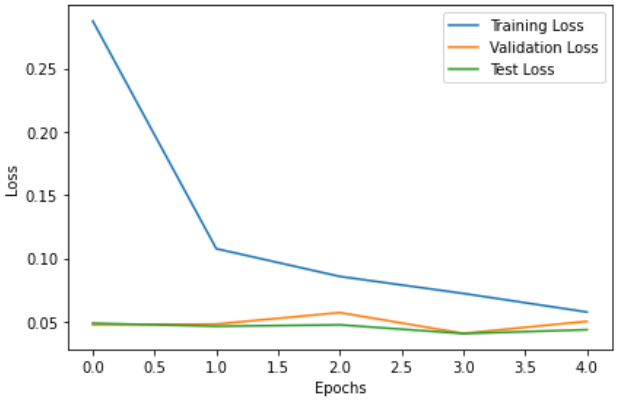
Plots:

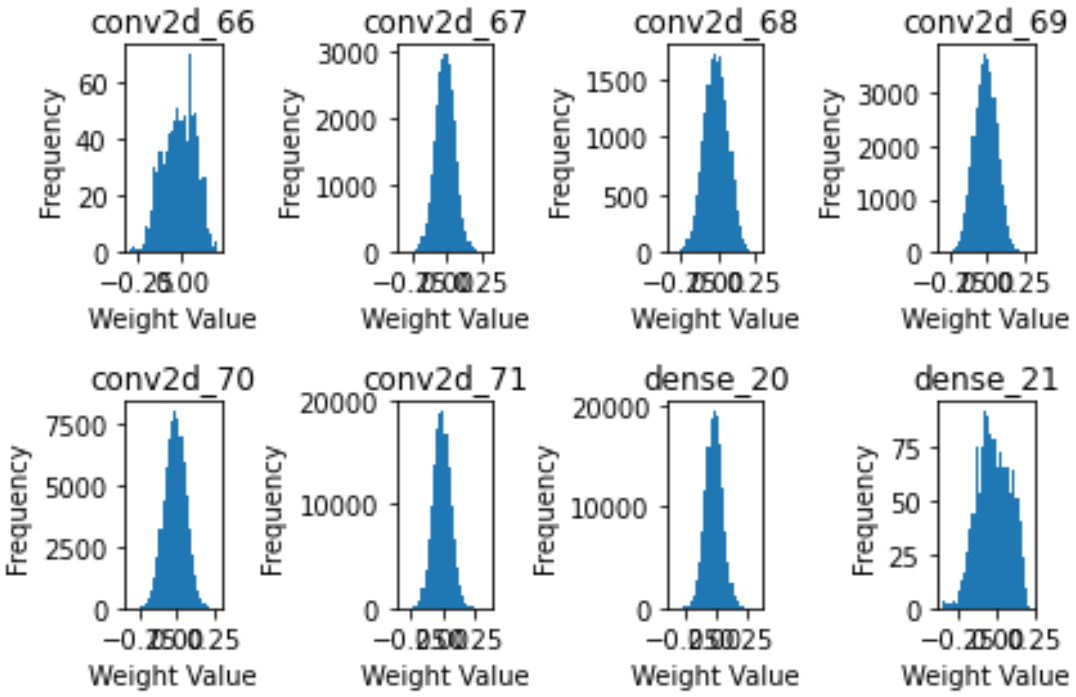
- The code includes plotting of training accuracy, validation accuracy, and test accuracy over epochs.
- Additional plots for training loss, validation loss, and test loss are included.
- Feature map, example figure after classification also shown

1. MNIST dataset

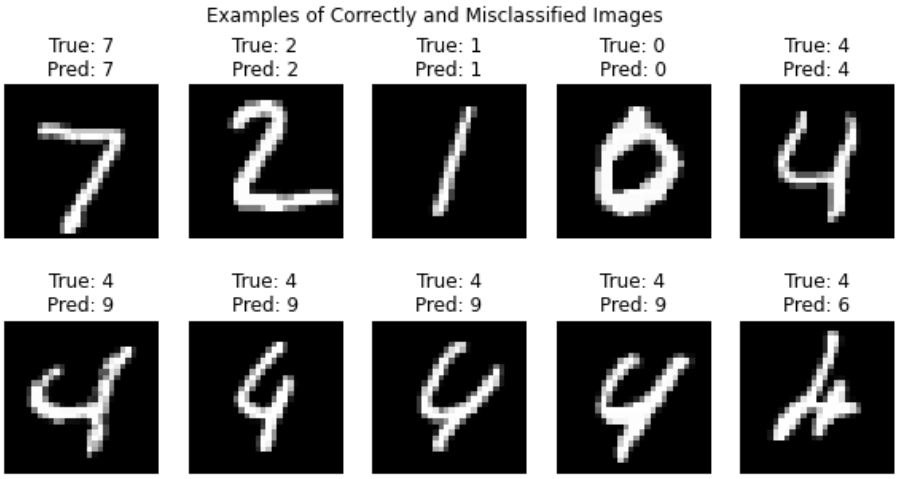
1-1





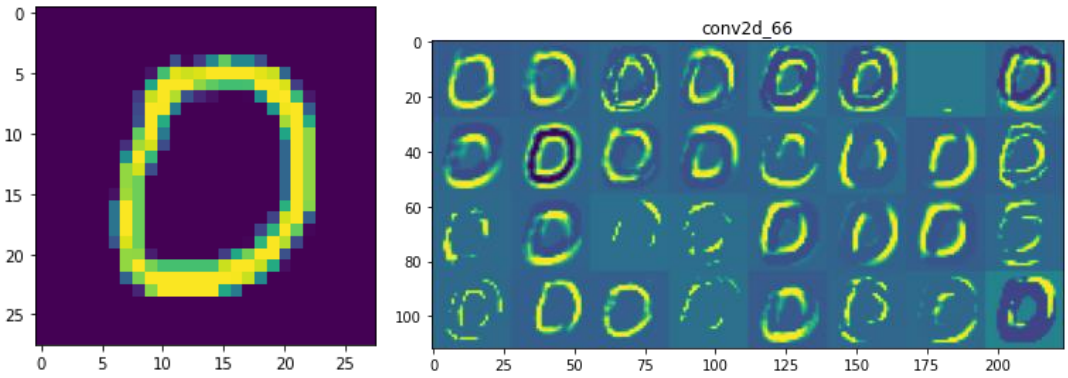


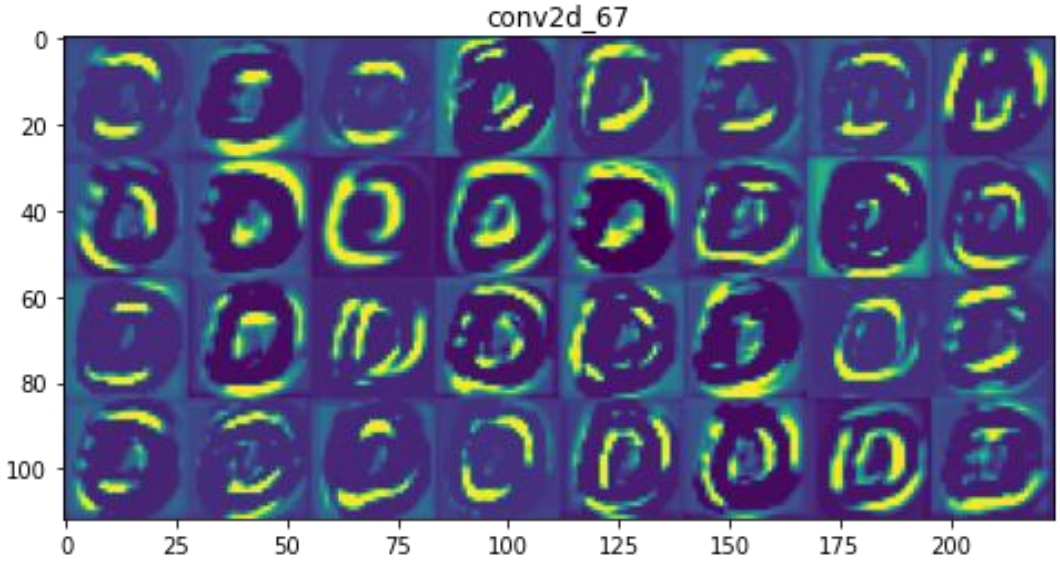
1-2

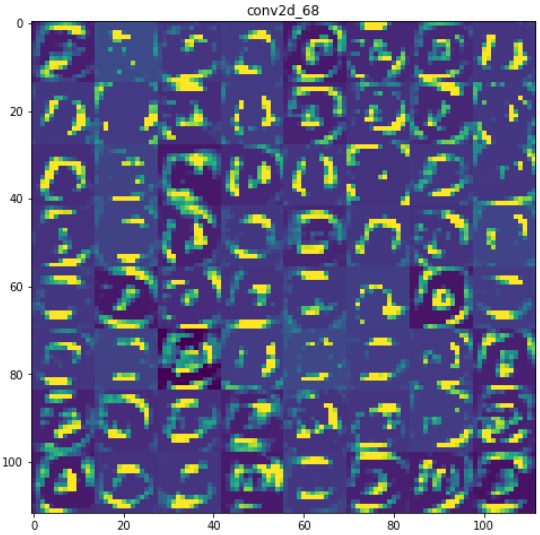


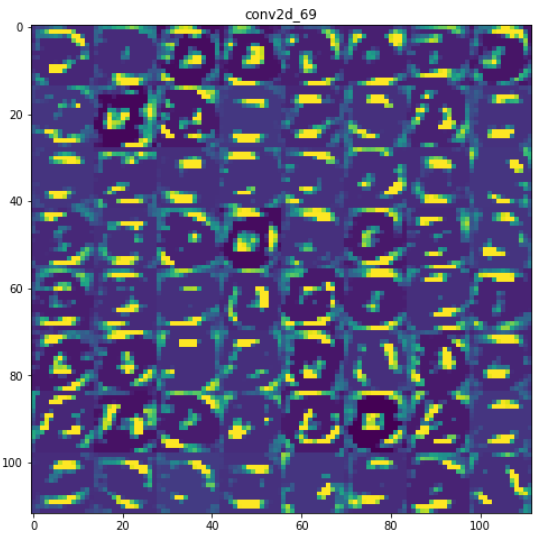
In the top row, where the model correctly predicts the class, the images display clear and distinctive patterns, showcasing the model's proficiency in unambiguous cases. Conversely, the bottom row highlights instances where the model misclassifies images, with titles indicating both the true and predicted labels. Analyzing these misclassifications is crucial for understanding the model's limitations and identifying specific challenges, such as ambiguous features or patterns that lead to errors. This insight enables iterative refinement of the model through strategies like fine-tuning, adjusting hyperparameters, or enhancing data diversity, ultimately contributing to improved overall performance and generalization.

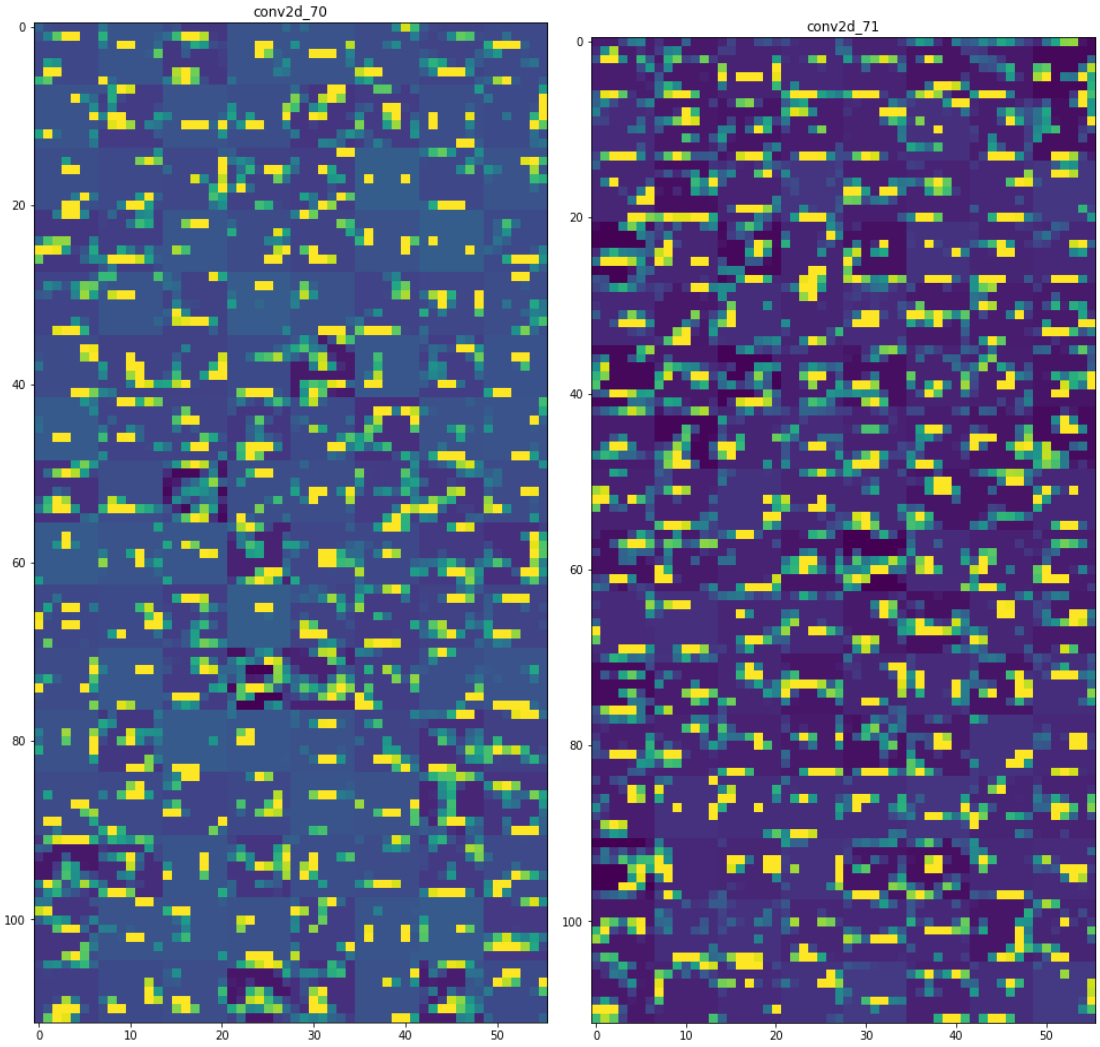
1-3




conv2d_67
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200


conv2d_68
0
20
40
60
80
100
0
20
40
60
80
100


conv2d_69
0
20
40
60
80
100
0
20
40
60
80
100

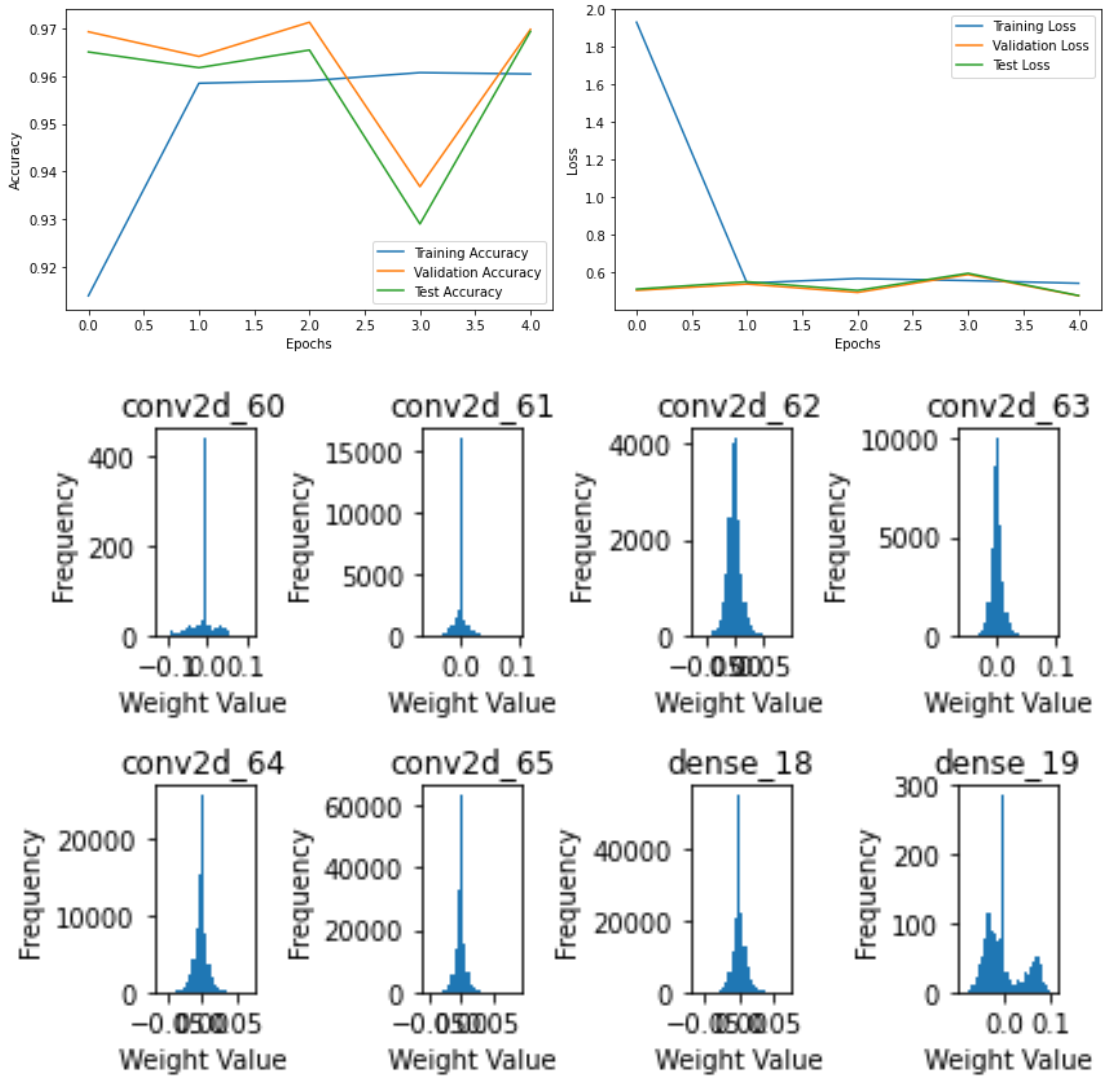


This first layer retains almost the full shape of the image, and also most of the information present in the image. As we go deeper into the network we can see that the activations become more complex and abstract. It starts encoding high-level features such as edges, curves and angles. Also as we go deeper we can see that many of our filters are not getting activated, which shows our model is reaching it's learning capacity.

In the context of a convolutional neural network (CNN), the initial layers, such as the first convolutional layer, are characterized by their ability to retain the full shape of the input image and preserve much of its detailed information. This layer acts as a feature extractor capturing low-level features, including basic patterns such as edges and color variations. As the network progresses into deeper layers, a shift occurs towards more complex and abstract representations. The activations in these deeper layers encode high-level

features like curves, angles, and other sophisticated patterns. This progression signifies the network's capacity to understand and represent intricate structures within the data. Notably, the observation that many filters in these deeper layers are not getting activated indicates that the model is reaching its learning capacity. This phenomenon reflects the specialization of filters, where certain filters respond selectively to specific features, contributing to the model's ability to discern diverse and intricate patterns in the input data.

1-4



The incorporation of L2 regularization into both convolutional neural network (CNN) layers and dense (fully connected) layers introduces several impactful effects on the model's training dynamics and generalization performance. L2

regularization serves as a valuable tool in mitigating overfitting by penalizing large weights in the model. This regularization technique encourages the learning of more compact and generalized representations, reducing sensitivity to noise in the training data. The resultant effect is a controlled model complexity that balances expressiveness with the prevention of overfitting.

In the context of CNN layers, L2 regularization influences the learning of spatial hierarchies and filters. It discourages individual filters from dominating the feature extraction process, promoting a diverse set of learned filters that capture a wider range of patterns. Similarly, in dense layers, L2 regularization shapes the learning of high-level abstractions, encouraging a more balanced combination of features.

Choosing the appropriate regularization strength is a critical aspect of this technique. While higher regularization strengths offer more aggressive weight penalties, lower values may not provide sufficient regularization. Striking the right balance through careful parameter tuning is essential for achieving optimal model performance. Additionally, L2 regularization is often employed in conjunction with other regularization methods, such as dropout, to achieve a nuanced trade-off between preventing overfitting and maintaining model expressiveness. Overall, the judicious use of L2 regularization contributes to improved generalization, reduced overfitting, and enhanced model robustness across diverse datasets.

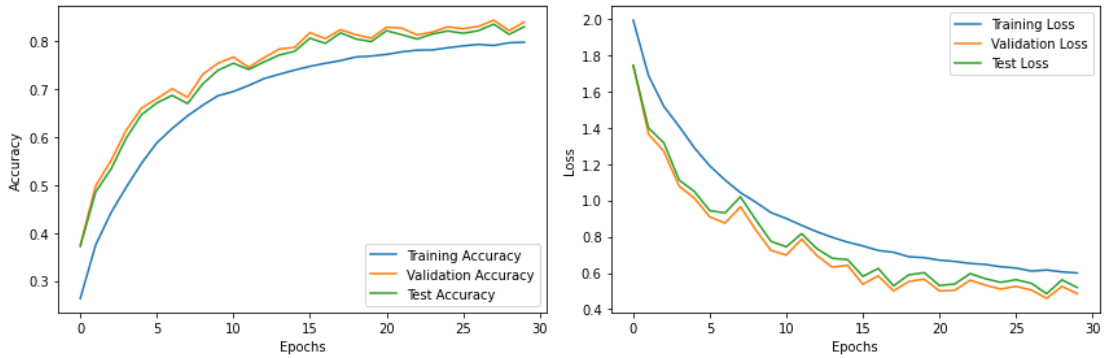
2. CIFAR-10

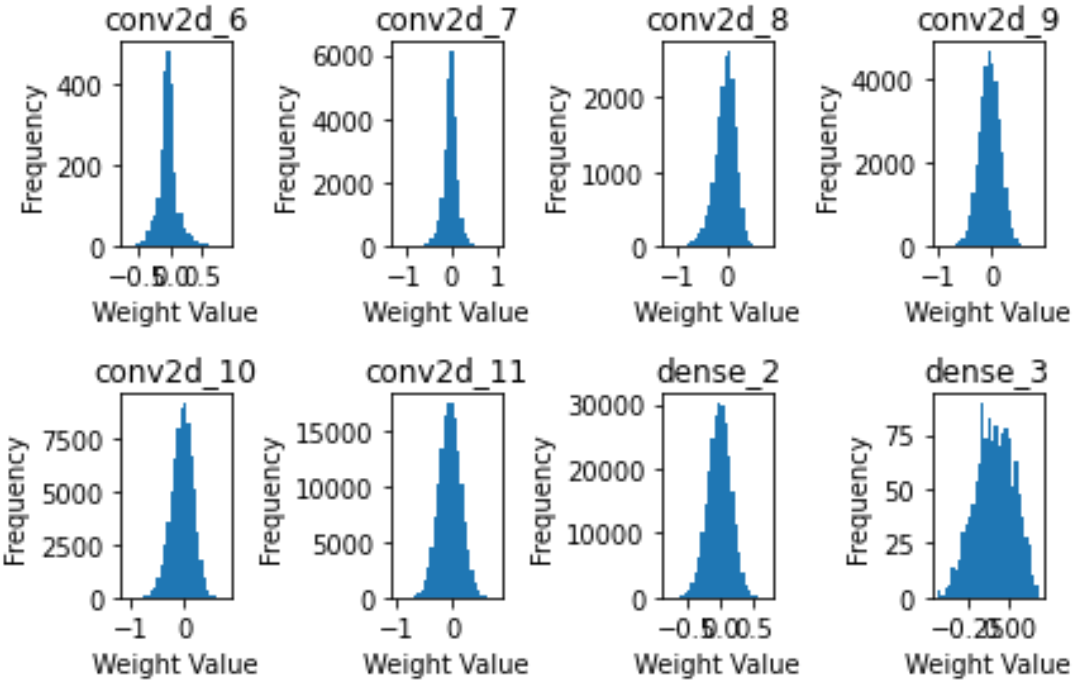
Preprocessing stage:

In the preprocessing and augmentation stages of the provided code, several steps were taken to prepare the input data for training the neural network. The pixel values of the image data were first normalized by dividing them by 255.0, scaling them to a range between 0 and 1. This normalization aids in the convergence of the training process. For the CIFAR-10 dataset, the data augmentation techniques were applied using the ImageDataGenerator from Keras. These included random rotations (up to 15 degrees), horizontal and vertical shifts (up to 10% of the image dimensions), shearing transformations (with a maximum shear angle of 0.2), random zooming (up to 20%), and horizontal flips. These augmentations introduce diversity into the training dataset, helping the neural network generalize better to variations it

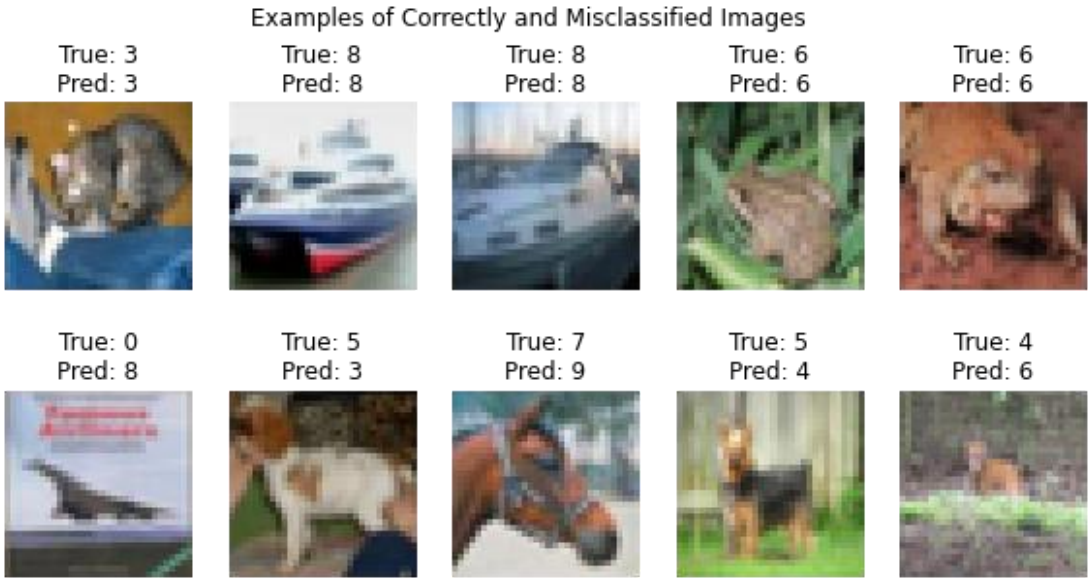
might encounter in real-world scenarios. The fill_mode='nearest' parameter in the augmentation configuration determined the strategy for filling in newly created pixels resulting from rotations or shifts, using the nearest existing pixel values. Overall, this preprocessing and augmentation pipeline enhances the model's robustness and improves its ability to handle variations in the input data during training.

2-1

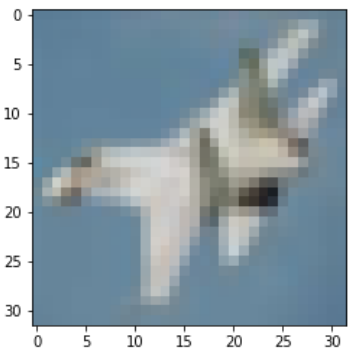


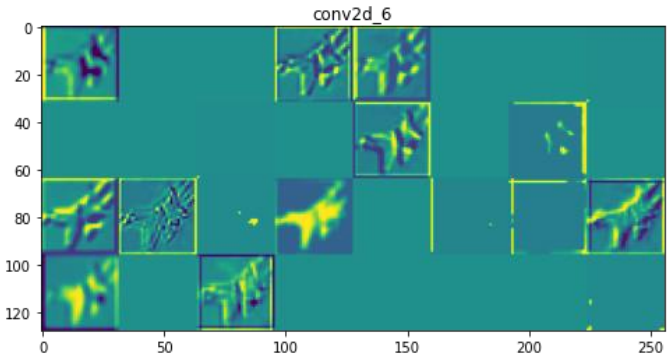


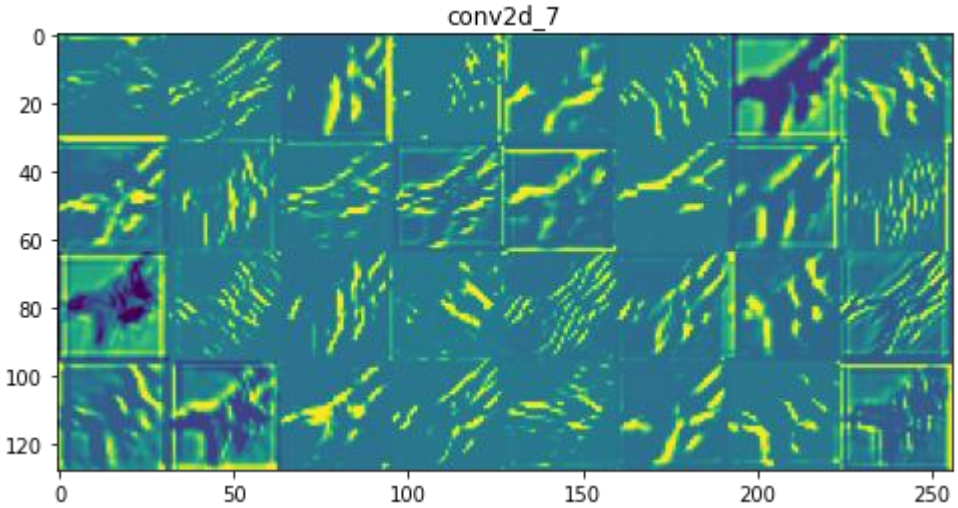
2-2

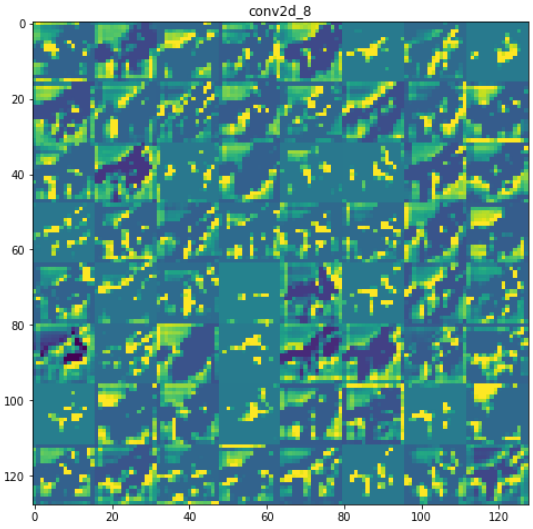

Examples of Correctly and Misclassified Images
True: 3
Pred: 3
True: 8
Pred: 8
True: 8
Pred: 8
True: 6
Pred: 6
True: 6
Pred: 6
True: 0
Pred: 8
True: 5
Pred: 3
True: 7
Pred: 9
True: 5
Pred: 4
True: 4
Pred: 6

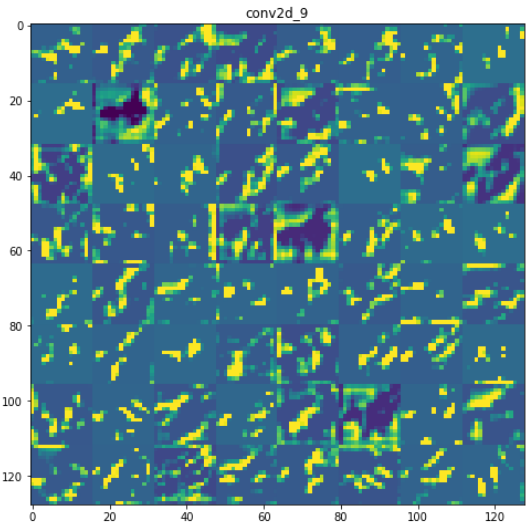
2-3

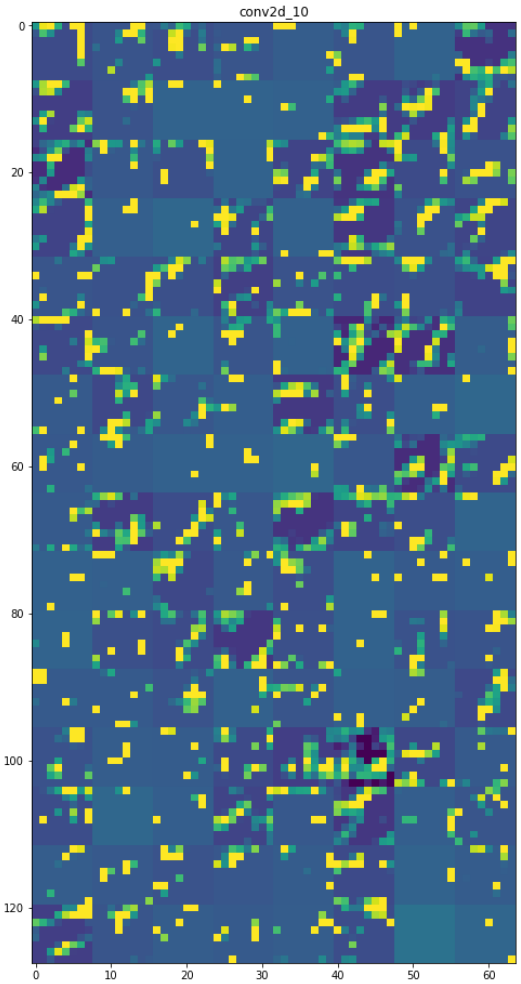


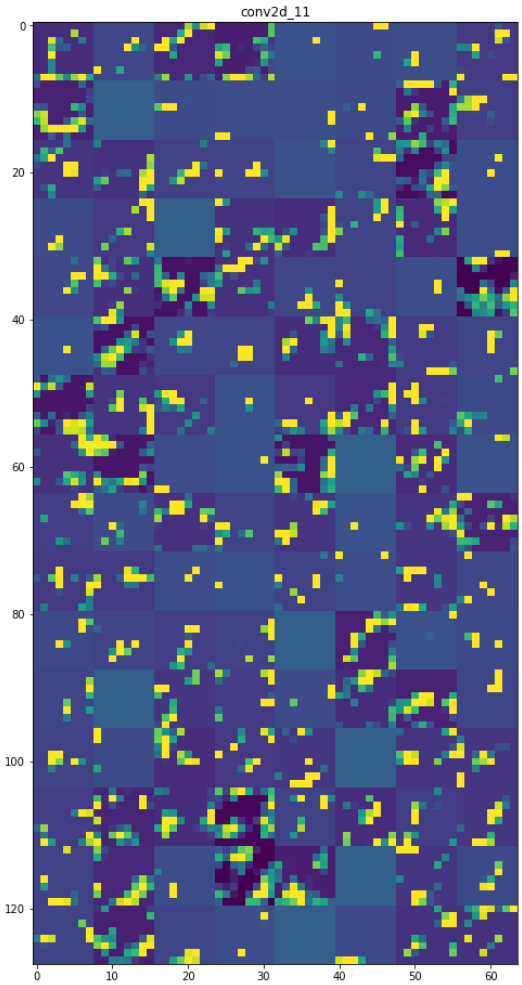

conv2d_6


conv2d_7


conv2d_8


conv2d_9


conv2d_10


conv2d_11