Task 1

- Test the network on the data set:

- How the network preforms on the whole dataset:

```
print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
Accuracy of the network on the 10000 test images: 55 %
```

- what are the classes that performed well, and the classes that did not perform well:

```
Accuracy for class: plane is 62.7 %
Accuracy for class: car   is 72.6 %
Accuracy for class: bird  is 27.2 %
Accuracy for class: cat   is 34.0 %
Accuracy for class: deer  is 39.2 %
Accuracy for class: dog   is 56.1 %
Accuracy for class: frog  is 70.7 %
Accuracy for class: horse is 66.4 %
Accuracy for class: ship  is 69.4 %
Accuracy for class: truck is 54.6 %
```

Description of Our Process

1. **Downloading and Loading the Dataset**:
    - We utilized the torchvision.datasets.CIFAR10 class to download and load the CIFAR-10 dataset.
2. **Transforming the Data**:
    - We applied a series of transformations to the data, including converting images to torch tensors and normalizing them.
3. **Defining the CNN Model**:
    - We constructed a simple Convolutional Neural Network (CNN) for image classification, which includes:
        - **Two Convolutional Layers**: Each followed by a ReLU activation function and a max-pooling layer.
        - **Three Fully Connected Layers**: The final layer outputs class probabilities for the 10 CIFAR-10 classes.
4. **Defining the Loss Function and Optimizer**:
    - We used Cross-Entropy Loss as the loss function, suitable for multi-class classification problems.
    - For optimization, we used Stochastic Gradient Descent (SGD) with a learning rate of 0.001 and a momentum of 0.9.
5. **Training the Network**:
    - We trained the network for 2 epochs. During each epoch, we:
        - Iterated over the training data in mini-batches.
        - Performed forward propagation to compute the outputs.
        - Calculated the loss using the defined loss function.
        - Executed backpropagation to compute the gradients.
        - Updated the network parameters using the optimizer.
6. **Evaluating the Network**:
    - We evaluated the trained network on the test data by:
        - Performing forward propagation on the test data to compute the outputs.
        - Determining the predicted classes by finding the class with the highest output probability.
        - Computing the overall accuracy.
        - Analyzing the accuracy for each class separately to understand the model's performance across different categories.

Results

- **Overall Accuracy**: The network achieved an accuracy of approximately 55% on the test set.
- **Class-wise Performance**: We observed that the model performs well on certain classes but struggles with others. This might be due to the varying complexity and visual similarity between different classes (e.g., boats and cars).
- **Visualization**: We visualized some sample images from the test set along with their predicted and true labels. This helped us understand the network's performance and identify instances where the model struggled, particularly with visually similar classes like ships and cars.

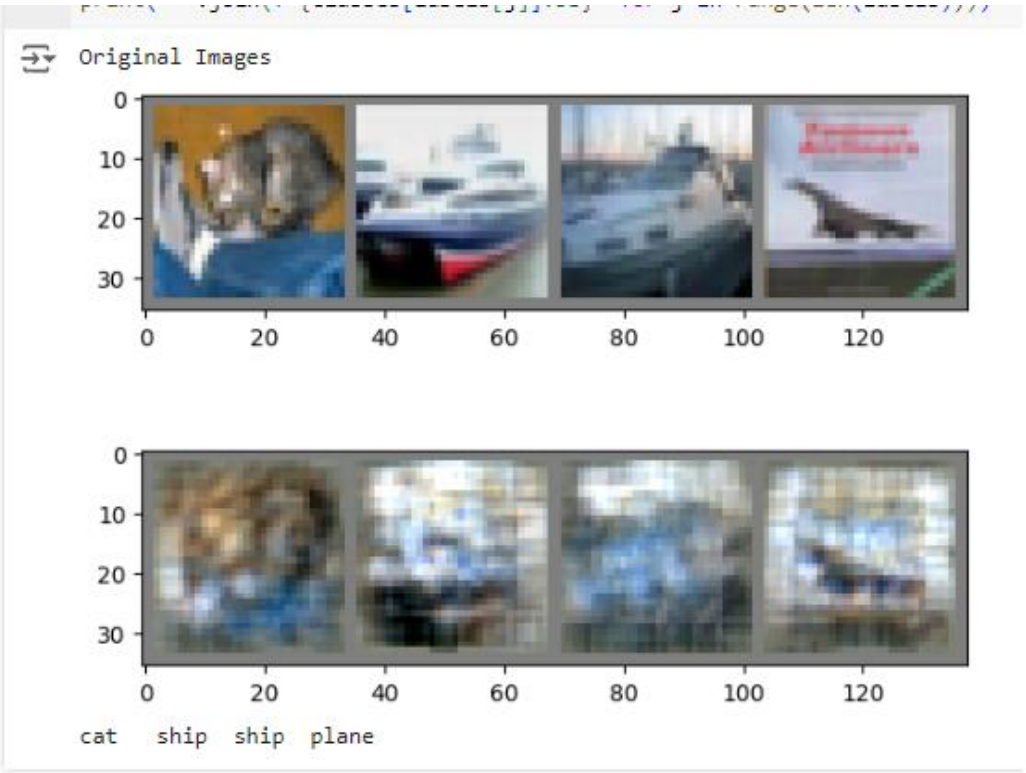
Task 2

- Report the classification error (in accuracy) of the test set

```
print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

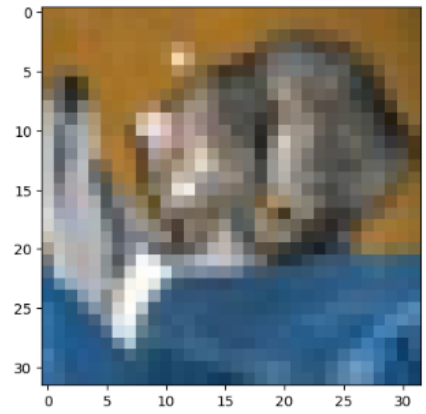
```
Accuracy of the network on the 10000 test images: 59 %
```

- show two-three examples of reconstructed images alongside the original images.

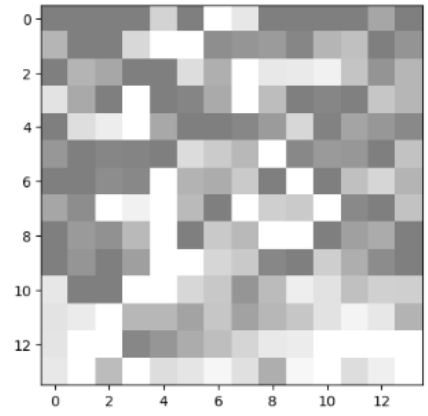


Task 3

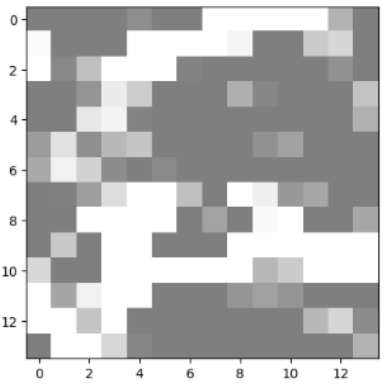
Original Images



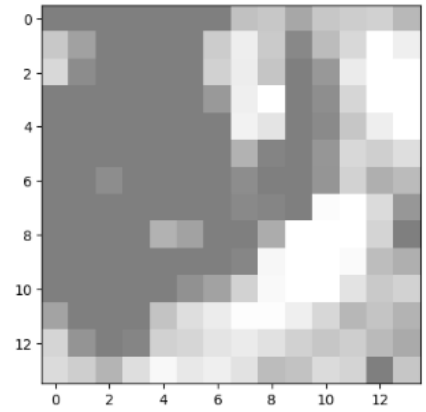
Reconstructed from channel 2 of layer 1



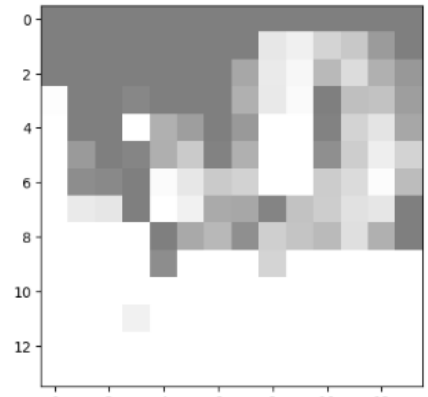
Reconstructed from channel 5 of layer 1



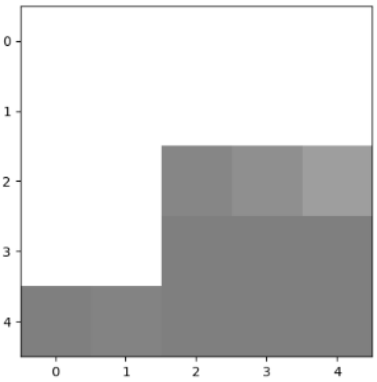
WARNING:matplotlib.image:Clipping input data to the valic
Reconstructed from channel 0 of layer 1



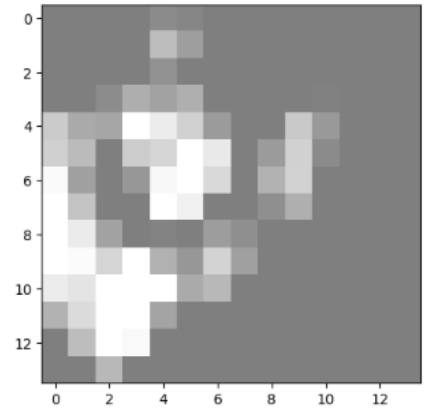
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 3 of layer 1



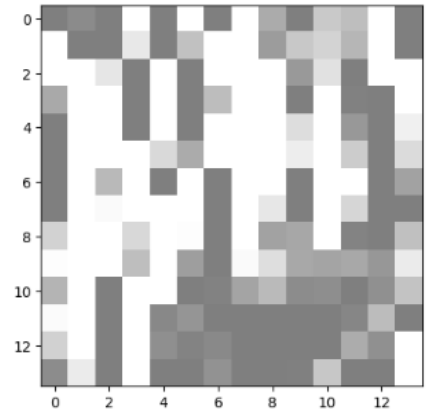
Reconstructed from channel 0 of layer 2
WARNING:matplotlib.image:Clipping input data to the valid



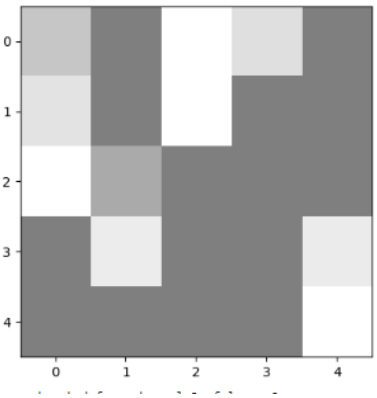
WARNING:matplotlib.image:Clipping input data to the valic
Reconstructed from channel 1 of layer 1



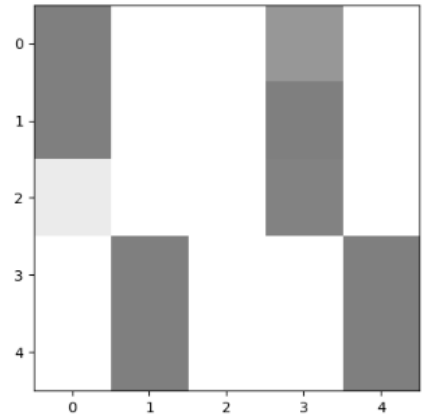
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 4 of layer 1



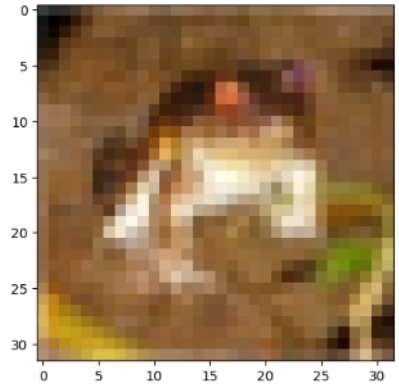
Reconstructed from channel 1 of layer 2
WARNING:matplotlib.image:Clipping input data to the valid



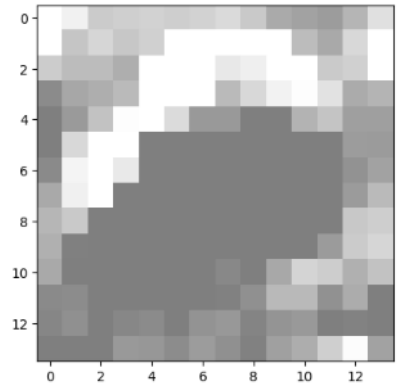
Reconstructed from channel 2 of layer 2
WARNING:matplotlib.image:Clipping input data to the valid



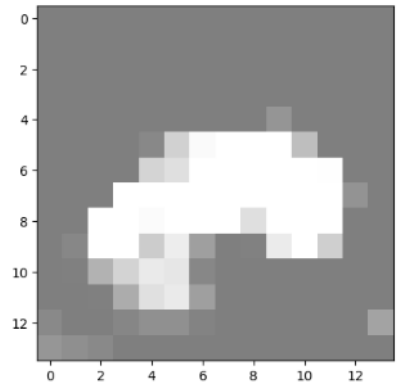
Original Images



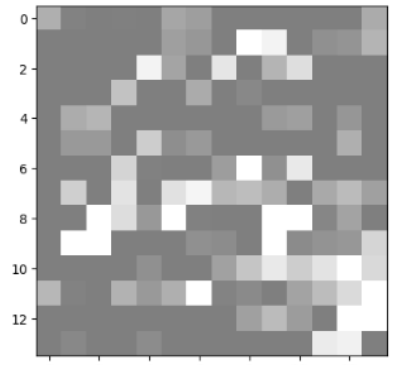
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 0 of layer 1



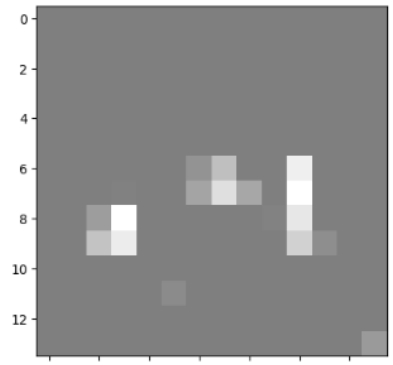
Reconstructed from channel 1 of layer 1WARNING:matplotlib.



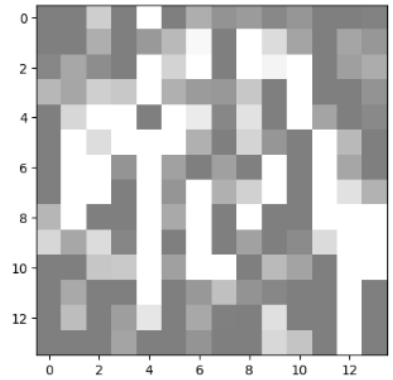
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 2 of layer 1



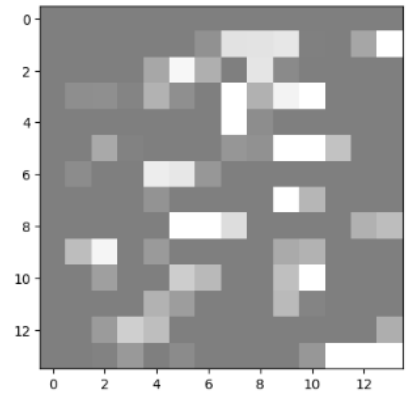
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 3 of layer 1



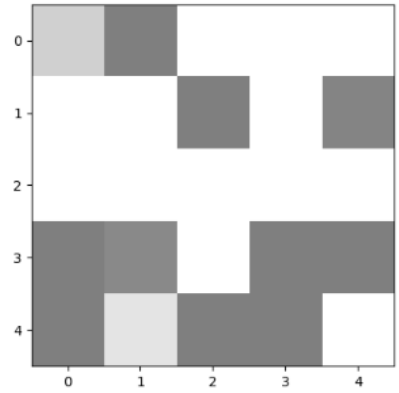
WARNING:matplotlib.image:Clipping input data to the valid
Reconstructed from channel 4 of layer 1



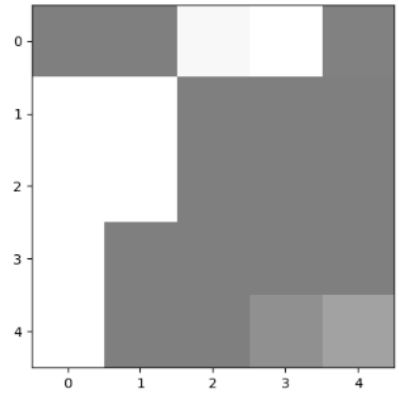
Reconstructed from channel 5 of layer 1



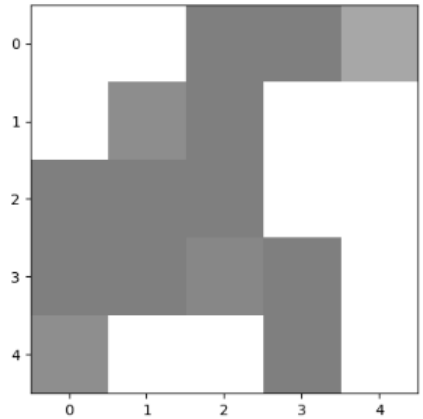
WARNING:matplotlib.image:Clipping input data to the valid ra
Reconstructed from channel 0 of layer 2



WARNING:matplotlib.image:Clipping input data to the valid ra
Reconstructed from channel 1 of layer 2



Reconstructed from channel 2 of layer 2
WARNING:matplotlib.image:Clipping input data to the valid



In the first layer, the network detects shapes and contours of objects in the image. In the second image, you can see that the first channel of the first layer clearly shows the contours of the frog. However, in the second layer, it is not currently possible to draw any definitive conclusions, likely because the pixels are too large in relation to the image.