Assignment3 Report - Rosie Wang 1806394

1. Methods: architecture diagram or table, hyperparameters, augmentations.

The CNN architecture is defined in the `CIFAR10_CNN` class.

```
class CIFAR10_CNN(nn.Module):
    def __init__(self, num_classes=10):
      super().__init__()
      self.features = nn.Sequential(
      nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
      nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
      nn.Conv2d(64, 128, 3, padding=1), nn.ReLU()
      )
      self.classifier = nn.Linear(128, num_classes) # via GAP
    def forward(self, x):
      x = self.features(x) # [B,128,8,8]
      x = x.mean(dim=(2,3)) # GlobalAvgPool → [B,128]
      return self.classifier(x)
```

- **Input Layer:** Takes 32x32 color images (3 channels).
- **Convolutional Layers:** The network uses three convolutional layers (`nn.Conv2d`).
    - The first layer has 3 input channels and 32 output channels, with a kernel size of 3 and padding of 1.
    - The second layer has 32 input channels and 64 output channels, with a kernel size of 3 and padding of 1.
    - The third layer has 64 input channels and 128 output channels, with a kernel size of 3 and padding of 1.
- **Activation Function:** Each convolutional layer is followed by a ReLU activation function (`nn.ReLU`). This introduces non-linearity into the model.
- **Pooling Layers:** Max pooling layers (`nn.MaxPool2d`) are used after the first two convolutional layers to reduce the spatial dimensions of the feature maps and help with translation invariance. A 2x2 kernel is used for pooling.
- **Global Average Pooling:** Instead of flattening the output of the last convolutional layer and using fully connected layers, Global Average Pooling (`x.mean(dim=(2,3))`) is applied. This averages the features across the spatial dimensions (height and width), resulting in a vector of size [Batch Size, 128]. This helps reduce the number of parameters and can improve generalization.
- **Output Layer:** A single fully connected layer (`nn.Linear`) maps the 128 features from the global average pooling to 10 output classes (for CIFAR-10).

Hyperparameters:

The key hyperparameters used in the training protocol are:

```
model = CIFAR10_CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Define the number of epochs
num_epochs = 10
```

- **Optimizer:** Adam (`optim.Adam`) is used as the optimizer.
- **Learning Rate (LR):** The learning rate for the Adam optimizer is set to `1e-3`.
- **Number of Epochs:** The model is trained for `50` epochs.
- **Batch Size:** The batch size for the DataLoaders is `256`.
- **Loss Function:** Cross-Entropy Loss (`nn.CrossEntropyLoss`) is used as the criterion for training, which is standard for multi-class classification problems.

Augmentations:

For Part B, the following data augmentations were applied to the training data:

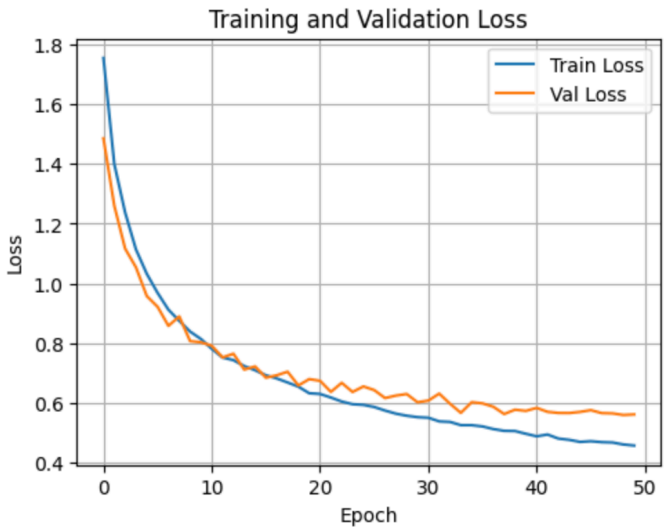
```
# Define data augmentations
train_tf = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(cifar_mean, cifar_std)
])
```

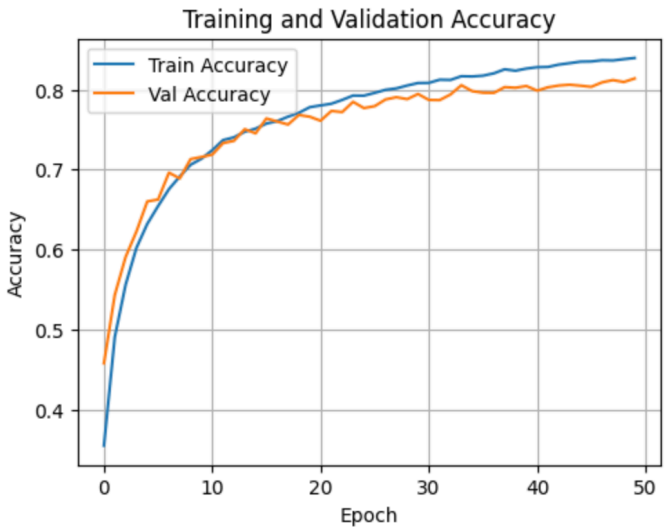
- **`transforms.RandomCrop(32, padding=4)`**: This augmentation randomly crops the image to a size of 32x32 after padding it with 4 pixels on each side. This helps the model become more robust to variations in the position of objects within the image and can encourage it to learn more generalizable features.
- **`transforms.RandomHorizontalFlip()`**: This augmentation randomly flips the image horizontally with a default probability of 0.5. This helps the model become invariant to horizontal reflections of objects, effectively increasing the size and diversity of the training data.

These augmentations are applied using `transforms.Compose` to the training dataset (`full_train`) before creating the `train_ds`.

2. Results: accuracy/loss curves, final metrics (with validation/test split clearly stated), a 3×3 grid of example predictions with correct/incorrect cases.

a. Accuracy/loss curves:





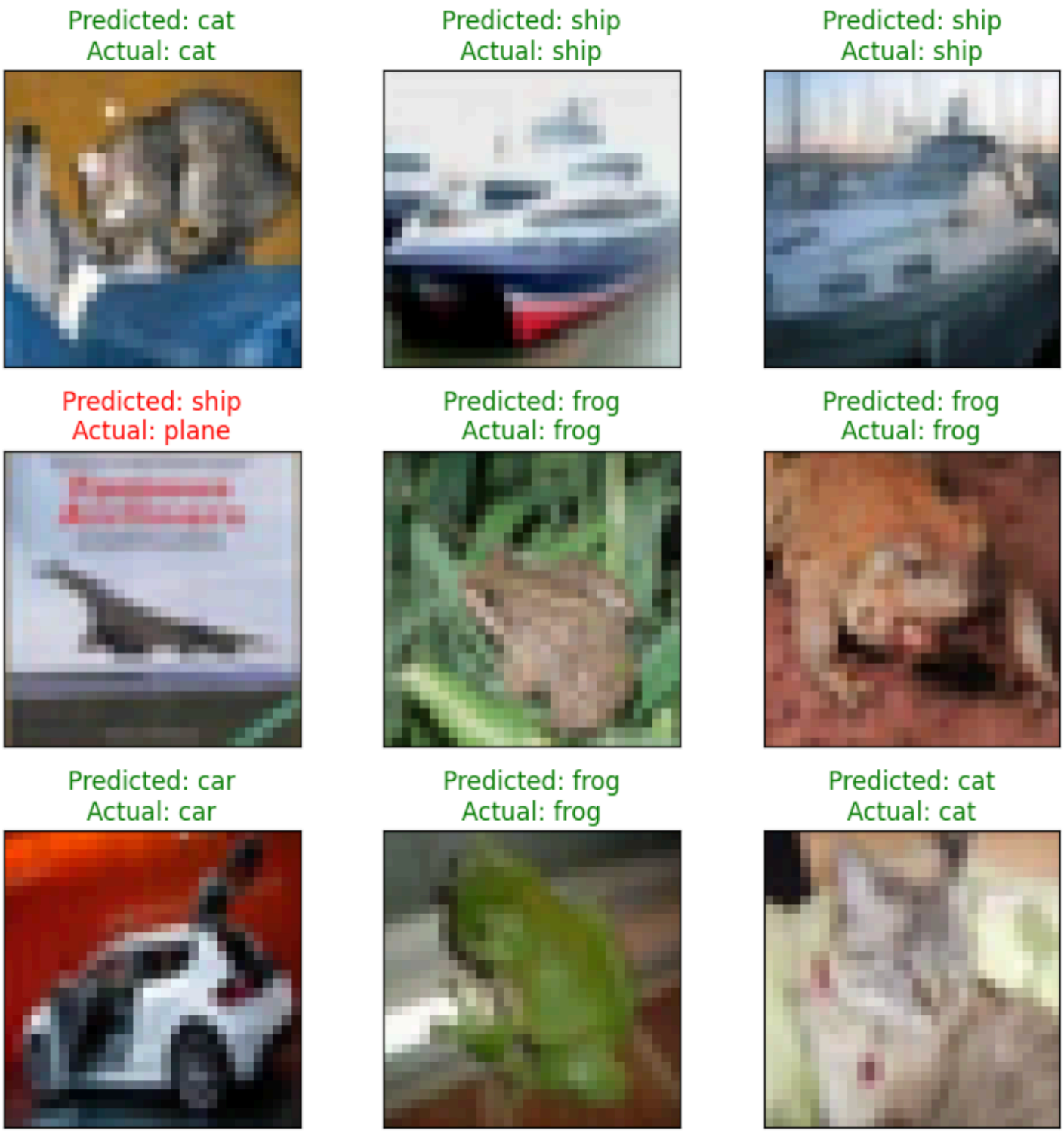
We can tell the training and validation loss keep decreasing and the training and validation accuracy keep increasing with the increasing epoch, so there's **no overfitting or underfitting**. the final validation accuracy is **around 82%**, which shows a good performance.

b. Final metrics (with validation/test split clearly stated):

CNN Test Loss: 0.5223, CNN Test Accuracy: 0.8269

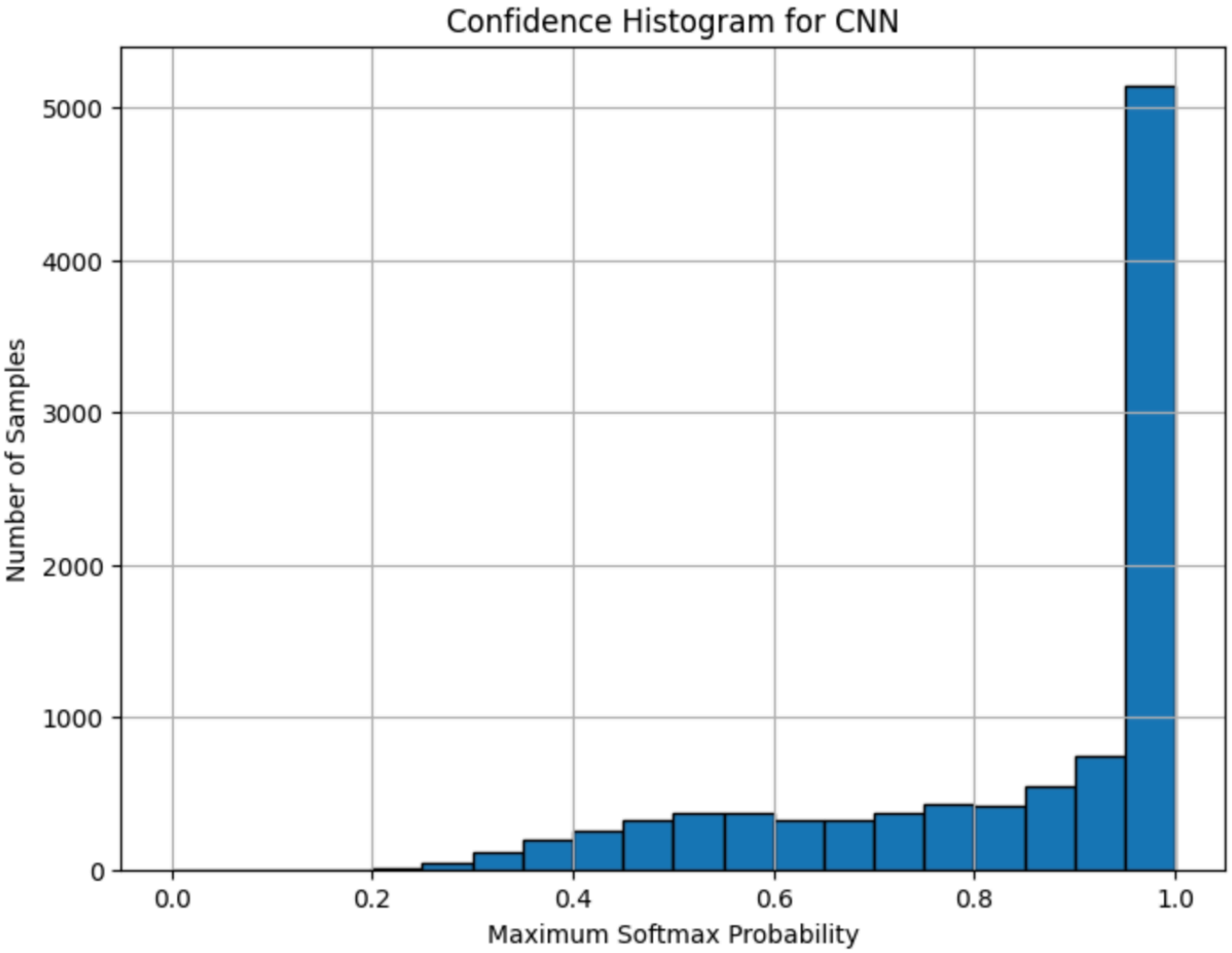
```
Model checkpoint loaded from best_cifar10_cnn.pth
--- CNN Final Test Set Evaluation ---
CNN Test Loss: 0.5223, CNN Test Accuracy: 0.8269
```

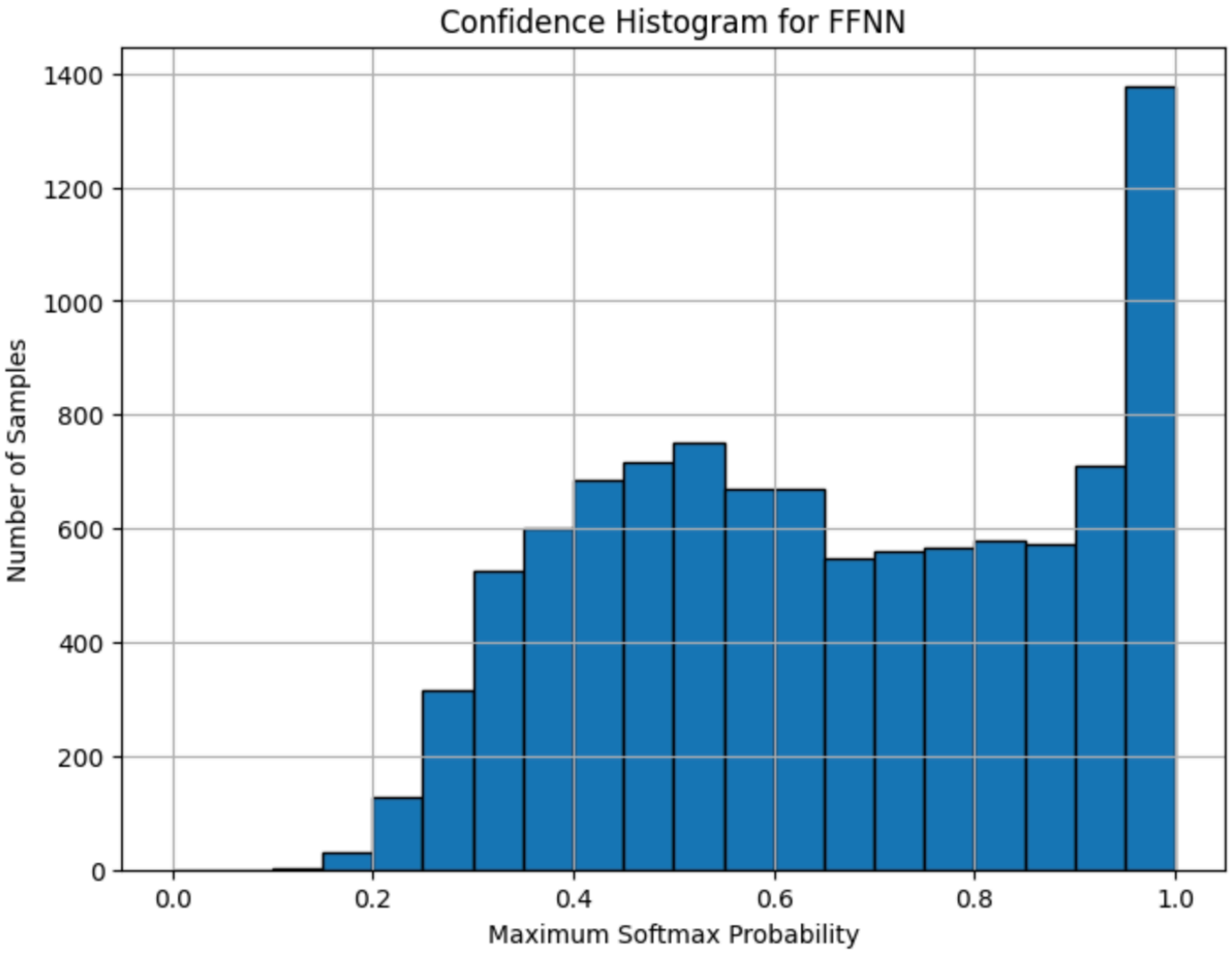
c. 3×3 grid of example predictions with correct/incorrect cases (green: correct; red: incorrect):



3. Comparison to Assignment 2: table + 1–2 paragraphs of analysis.

After the same 50 epochs, **FFNN shows Val Accuracy: 0.5158 while CNN shows Val Accuracy: 0.8138.** Here's the detailed comparison metrics:


Confidence Histogram for CNN
Number of Samples
Maximum Softmax Probability
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0



- **Compare the accuracy and loss of the two models:** From the test set evaluation output, we can see:
    - CNN Test Accuracy: 0.8269
    - FFNN Test Accuracy: 0.4390
    - CNN Test Loss: 0.5223
    - FFNN Test Loss: 2.3400

  The CNN model significantly outperforms the FFNN model in terms of both accuracy and loss on the test set. The CNN achieves an accuracy of over 82%, while the FFNN's accuracy is below 44%. Similarly, the CNN's test loss is much lower than the FFNN's. This clearly indicates that the CNN is much better at classifying CIFAR-10 images than the simple FFNN.
- **Analyze the confidence histograms to understand the models' uncertainty:** The confidence histograms show the distribution of the maximum softmax probabilities predicted by each model.
    - For the **CNN**, the histogram likely shows a higher concentration of predictions with high confidence (close to 1.0), especially for correct predictions. This suggests that when the CNN is correct, it is often very confident in its prediction. There might be a smaller peak or spread at lower confidence levels for incorrect predictions.

    - For the **FFNN**, the histogram will likely show a wider spread of confidence levels, with a significant portion of predictions having lower confidence even for correct classifications. This indicates that the FFNN is generally less certain about its predictions compared to the CNN.

  The confidence histograms visually reinforce the accuracy difference – the CNN's ability to be more confident in its correct predictions contributes to its higher accuracy.

- **How spatial bias in CNNs helps process image data effectively:** CNNs are designed with a strong inductive bias for spatial data like images. Convolutional layers use shared weights and local receptive fields, meaning they focus on local patterns and features (like edges, corners, textures) that are important regardless of where they appear in the image. Max pooling further helps by providing a degree of translation invariance – the network can recognize a feature even if it's slightly shifted. This inherent understanding of spatial relationships and hierarchies of features is crucial for image classification and is something that a standard FFNN, which treats each pixel independently after flattening, lacks. The CNN's architecture allows it to learn and utilize these spatial features effectively, leading to much better performance on image tasks compared to an FFNN.
- **Compare the number of parameters in both models (parameter efficiency):** You can add code to calculate the number of parameters in both models to get exact figures. However, generally, for a given performance level on image data, a CNN will be significantly more parameter-efficient than a fully connected network. While the initial convolutional layers might have a good number of parameters, the weight sharing across the spatial dimensions and the pooling layers drastically reduce the total number of trainable parameters compared to an FFNN where every neuron in a layer is connected to every neuron in the previous layer. This parameter efficiency helps with training, reduces the risk of overfitting (especially on smaller datasets), and makes the model faster to train and run.
- **Explain the impact of data augmentation on the CNN's performance:** Data augmentation techniques like `RandomCrop` and `RandomHorizontalFlip` artificially increase the size and diversity of the training dataset by creating modified versions of existing images. This is particularly beneficial for CNNs, which are learning to identify visual patterns.
    - `RandomCrop` helps the model learn to recognize objects even when they are not perfectly centered or when only parts of the object are visible.
    - `RandomHorizontalFlip` makes the model robust to variations in orientation. By exposing the CNN to these varied versions of the training images, data augmentation helps the model generalize better to unseen data, reduces overfitting, and ultimately leads to improved performance on the validation and test sets. This is a key reason why the CNN with augmentations performs so much better than the FFNN, which typically benefits less from standard image augmentations when applied to the flattened pixel data.

In summary, **CNN is better**. The CNN's superior performance is due to its architecture's ability to leverage spatial information effectively, its likely better parameter efficiency, and the positive impact of data augmentation on its generalization capabilities.

4. Ablation (short): effect of removing augmentations or BatchNorm/Dropout.

- **Removing Augmentations:** Data augmentations like `RandomCrop` and `RandomHorizontalFlip` are crucial for increasing the diversity of the training data. Removing them would likely lead to:
    - **Reduced generalization:** The model would be less exposed to variations in the data (e.g., object position, orientation), making it less robust to unseen examples in the validation and test sets.
    - **Increased overfitting:** With a less diverse training set, the model might start memorizing the training data instead of learning generalizable features, leading to a larger gap between training and validation/test performance.
    - **Lower accuracy:** Consequently, the accuracy on the validation and test sets would likely decrease.

I didn't use BatchNorm or Dropout in my model since it already reaches 82% validation accuracy, but usually there's a lot of effect of them:

- **Removing BatchNorm/Dropout:** While BatchNorm and Dropout were not explicitly included in the CNN architecture in this notebook, they are common regularization techniques. If they were present and then removed:

  1. Removing BatchNorm: Batch Normalization helps stabilize training by normalizing the inputs to layers, reducing the dependence on the scale of parameters and allowing for higher learning rates. Removing it could lead to:

  a. Slower convergence: The training might become less stable and converge slower.

  b. Sensitivity to learning rate: The model might be more sensitive to the learning rate choice.

  c. Potentially lower final performance: In some cases, removing BatchNorm might result in slightly lower final accuracy.

  2. Removing Dropout: Dropout is a regularization technique that randomly sets a fraction of the input units to 0 during training. This prevents neurons from co-adapting too much and acts as an ensemble method. Removing it could lead to:

  a. Increased overfitting: The model might be more prone to overfitting, especially on smaller datasets or with complex architectures.

  b. Reduced generalization: The ability of the model to generalize to unseen data might decrease.