Name: **Sai Aravind Reddy Katteragandla**
Email: skatteragandla@csuchico.edu

CSCI 611 - 01 Applied Machine Learning Spring 2025

Assignment 2: CNN

Introduction: This project explores image processing techniques and Convolutional Neural Networks (CNNs) for image classification. In Part 1, we used various image filtering techniques such as edge detection, blurring, and scaling to preprocess images. In Part 2, we developed and trained a CNN model using the CIFAR-10 dataset to classify images into 10 categories. The study focuses on feature extraction through image processing and how CNNs learn hierarchical features for accurate classification.

Objectives:

Part 1: **Image Processing**

- Applied **edge detection, blurring, and scaling** as preprocessing steps to enhance feature extraction for deep learning models.
- Implemented convolution Edge Detection operations with different kernels for filtering.

Part 2: **CNN Implementation**

- Built a deep learning model using PyTorch.
- Trained the model on the CIFAR-10 dataset to classify images into 10 categories.
- Performance evaluation and result visualization implemented.

CNNs have many applications in computer vision, which include object detection and image classification. In this project, we have applied various convolutional layers, activation functions, and pooling layers in sequence to extract features from images and classify them effectively.

Implementation, training procedure, and evaluation of the model are presented in this report.

Overview:

Part 1: Image Processing / Filtering

Image processing is a critical step in the enhancement of visual features before sending images to CNN. We employed convolution-based filtering techniques, including edge detection for boundary detection, blurring for noise removal, and scaling for resolution change. Such preprocessing steps allow CNNs to learn informative patterns while removing undesired variations.

Techniques Used:

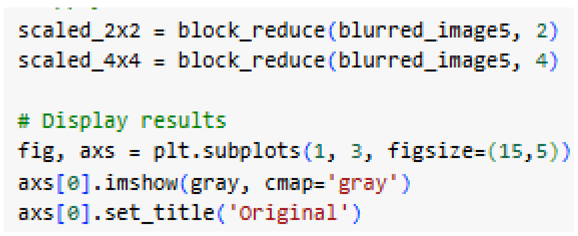
1. Edge Detection: Edge detection enhances object boundaries by identifying areas of sharp intensity changes. We applied the Sobel filter, which detects horizontal and vertical edges.

```
sobel_x = np.array([[-1, -2, -1],
                     [ 0,  0,  0],
                     [ 1,  2,  1]])

sobel_y = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]])

# Apply Sobel filters
sobel_x_filtered = cv2.filter2D(gray, -1, sobel_x)
sobel_y_filtered = cv2.filter2D(gray, -1, sobel_y)
```

2. Blurring (Smoothing): Blurring eliminates noise and smooths the image by averaging pixel values. In this case, we employed a 5×5 averaging filter to blur image details before edge detection. Preprocessing in this way helps eliminate false edges by removing high-frequency noise.

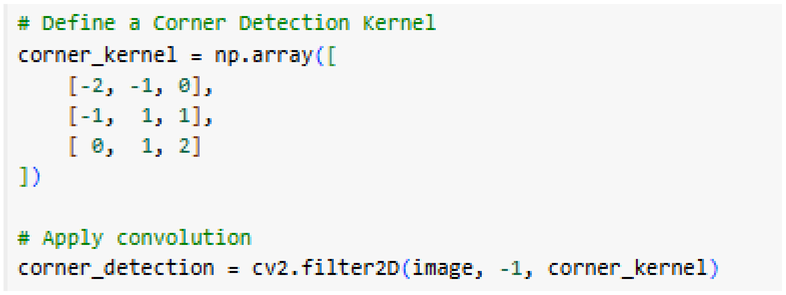
```
vertica = np.array([[ -1, 0, 1],
                    [ -1, 0, 1],
                    [ -1, 0, 1]])

# Step 2: Apply a 5x5 blurring filter
blurred_img = cv2.filter2D(gray, -1, np.ones((5,5))/25.0)

# Step 3: Apply vertical edge detection on blurred image
edge_blurred = cv2.filter2D(blurred_img, -1, vertica)

# Step 4: Apply vertical edge detection on original (for comparison)
edge_original = cv2.filter2D(gray, -1, vertica)
```

3. Scaling: Scaling is used to change image size without losing significant details. We used 2x2 and 4×4 scaling techniques.



```
scaled_2x2 = block_reduce(blurred_image5, 2)
scaled_4x4 = block_reduce(blurred_image5, 4)

# Display results
fig, axs = plt.subplots(1, 3, figsize=(15,5))
axs[0].imshow(gray, cmap='gray')
axs[0].set_title('Original')
```

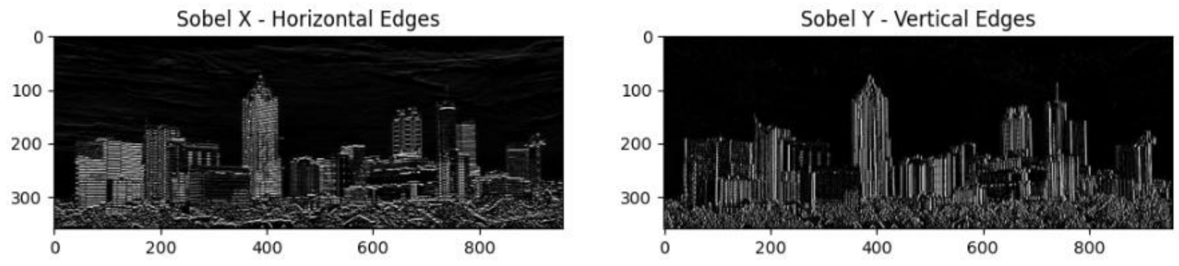
4. Corner Detection: Corner detection helps to locate points in an image where the intensity has a sharp change in more than one direction. We apply here a bespoke Sobel-like kernel via convolution to identify corner-like features. Then the output is binary thresholder with a high-contrast black-and-white presentation of detected corners. This helps to extract significant structural information from the image.



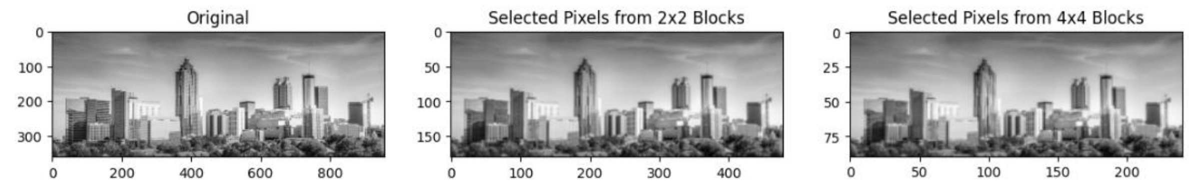
```
# Define a Corner Detection Kernel
corner_kernel = np.array([
    [-2, -1, 0],
    [-1,  1, 1],
    [ 0,  1, 2]
])

# Apply convolution
corner_detection = cv2.filter2D(image, -1, corner_kernel)
```

Results:

1. **Edge Detection**

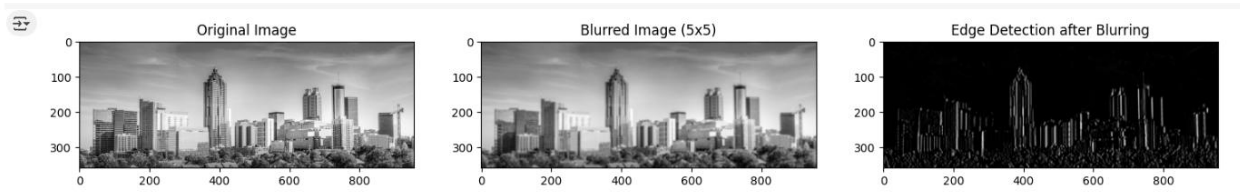


2. **Image Scaling**

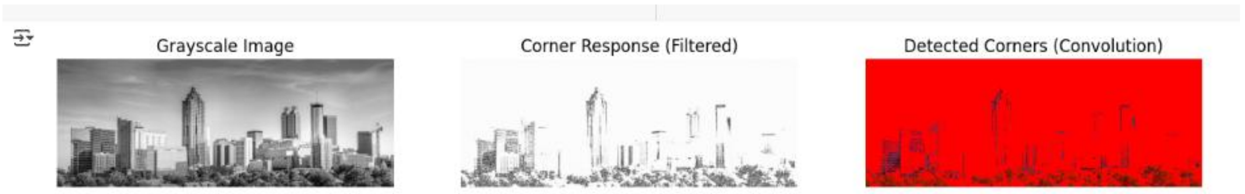


Original
Selected Pixels from 2x2 Blocks
Selected Pixels from 4x4 Blocks
0
100
200
300
200
400
600
800
50
150
25
75

3. **Series of filters: first one that blurs the image (takes an average of pixels), and then one that detects the edges.**



4. **Corner Detection (Convolution-Based Corner Detection Kernel) - To** detect steep intensity transitions at feature points in the image



Part 2: CNN Implementation

Dataset & Preprocessing

The CIFAR-10 dataset is a collection of 60,000 labeled images divided into 50,000 training images and 10,000 test images in 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Images are 32×32 pixels with three color channels (RGB), making it a de facto benchmark for image classification tasks

For enhancing model performance, we employed data normalization, normalizing the pixel values to the range -1 to 1 so that all the features contribute equally to learning. It prevents issues arising from varying intensities of pixels and accelerates convergence.

Training & Evaluation

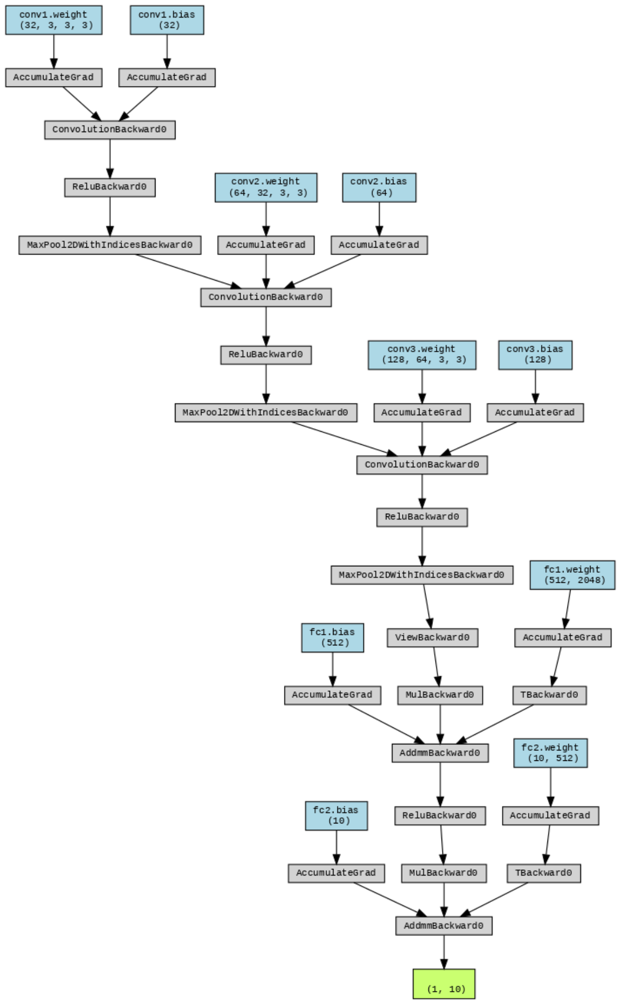
Our CNN model was trained with the gradient descent optimization process, minimizing the classification error over a few epochs. The following training hyperparameters were used

- Loss Function: CrossEntropyLoss
- Optimizer: SGD with momentum (0.9)
- Batch Size: 64
- Epochs: 10
- Learning Rate: 0.01

The model was trained using gradient descent, with each epoch updating the model weights to minimize loss.

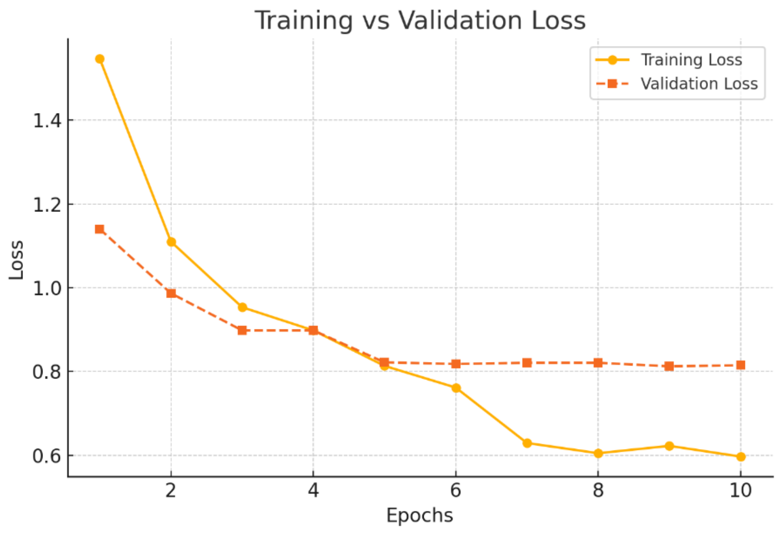
CNN Architecture Diagram

The architecture of the model implemented is depicted below. The model consists of three convolutional layers with max-pooling, dropout, and two fully connected layers for classification



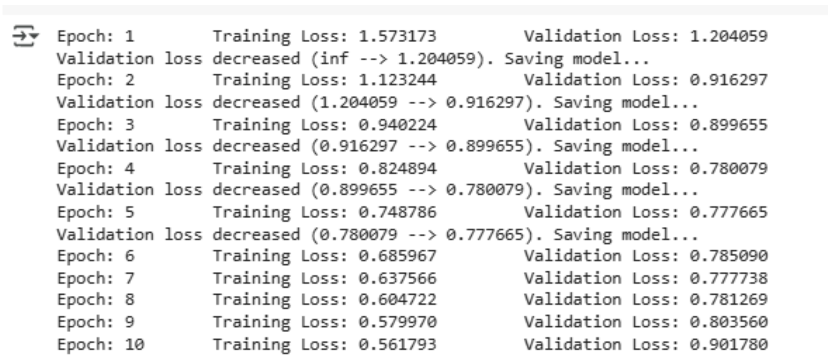
Training & Validation Loss:

The following graph plots the training and validation loss of over 10 epochs. The decreasing validation loss shows that the model is learning patterns from the CIFAR-10 dataset successfully



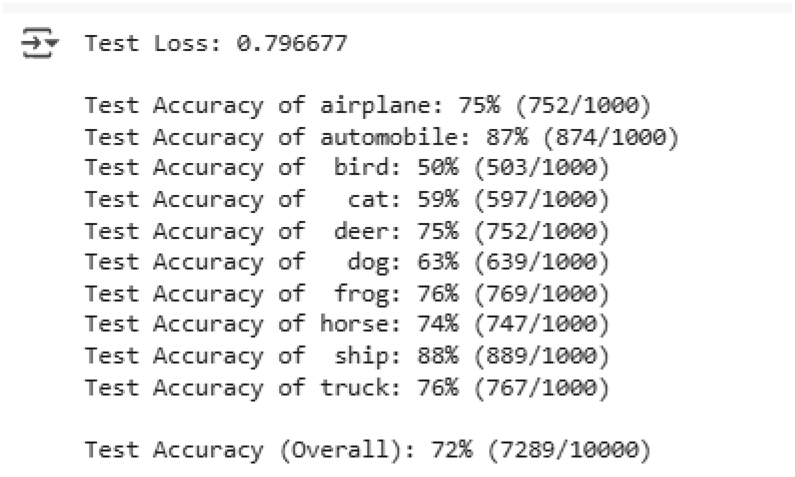
Results & Analysis:

- **Train the Network**



```
Epoch: 1        Training Loss: 1.573173         Validation Loss: 1.204059
Validation loss decreased (inf --> 1.204059). Saving model...
Epoch: 2        Training Loss: 1.123244         Validation Loss: 0.916297
Validation loss decreased (1.204059 --> 0.916297). Saving model...
Epoch: 3        Training Loss: 0.940224         Validation Loss: 0.899655
Validation loss decreased (0.916297 --> 0.899655). Saving model...
Epoch: 4        Training Loss: 0.824894         Validation Loss: 0.780079
Validation loss decreased (0.899655 --> 0.780079). Saving model...
Epoch: 5        Training Loss: 0.748786         Validation Loss: 0.777665
Validation loss decreased (0.780079 --> 0.777665). Saving model...
Epoch: 6        Training Loss: 0.685967         Validation Loss: 0.785090
Epoch: 7        Training Loss: 0.637566         Validation Loss: 0.777738
Epoch: 8        Training Loss: 0.604722         Validation Loss: 0.781269
Epoch: 9        Training Loss: 0.579970         Validation Loss: 0.803560
Epoch: 10       Training Loss: 0.561793         Validation Loss: 0.901780
```

- **Test the Trained Network**



```
Test Loss: 0.796677

Test Accuracy of airplane: 75% (752/1000)
Test Accuracy of automobile: 87% (874/1000)
Test Accuracy of  bird: 50% (503/1000)
Test Accuracy of   cat: 59% (597/1000)
Test Accuracy of  deer: 75% (752/1000)
Test Accuracy of   dog: 63% (639/1000)
Test Accuracy of  frog: 76% (769/1000)
Test Accuracy of horse: 74% (747/1000)
Test Accuracy of  ship: 88% (889/1000)
Test Accuracy of truck: 76% (767/1000)

Test Accuracy (Overall): 72% (7289/10000)
```

- **Visualize Sample Test Results**

Test Accuracy

After completing training, the model was evaluated on the CIFAR-10 test set of 10,000 unseen images. The model achieved total test accuracy of 72%, indicating a good performance in image classification tasks

Class-wise Performance

Class	Accuracy
Airplane	75%
Automobile	87%
Bird	50%
Cat	59%
Deer	75%
Dog	63%
Frog	74%

Horse	76%
Ship	88%
Truck	76%

The **highest** classification accuracy was reported in items such as **vehicles (87%)** and **ships (88%),** in which the shape and characteristics are constant. **Birds (50%)** had **the poorest accuracy**, which could be due to large intra-class variability and closeness to other classes

Confusion Matrix

The **confusion matrix** provides insights into class-wise performance, highlighting misclassifications between visually similar categories. For example, animals such as cats and dogs often show **overlapping features**, leading to classification errors

Conclusion

Summary of Findings

- CNN achieved **72% test accuracy** on CIFAR-10.
- The model performed well on structured objects (e.g., cars, ships) but poorly on animals (e.g., cats, birds).

Potential Improvements

To further enhance accuracy, we can:

- Make CNN more complex (add additional layers).
- Employ Adam optimizer instead of SGD.
- Employ Data Augmentation (flipping, cropping, rotations).
- Train for more epochs (20+).

Key Takeaways

- Edge detection & filtering are useful in preprocessing.
- CNNs automatically learn hierarchical features from images.
- More information & improved architecture enhances performance.