Name: SOUMALYA MONDAL
Roll No: M24DE2031

Report on CNN Pooling Strategies and Feature Map Visualization

1. Introduction

This report documents the implementation of a Convolutional Neural Network (CNN) to compare pooling strategies, visualize feature maps, and evaluate model performance on both the MNIST and Caltech-101 datasets. The analysis includes:

a. Comparison of Max Pooling and Average Pooling on model performance.
b. Feature map visualization after the first and second convolutional layers for both datasets.
c. Results and conclusions from the experiments using different pooling strategies and configurations of loss functions and optimizers.

2. Dataset Preparation

MNIST and Caltech dataset have been used here and data preprocessing was performed for both the dataset to prepare the data for training the CNN model.

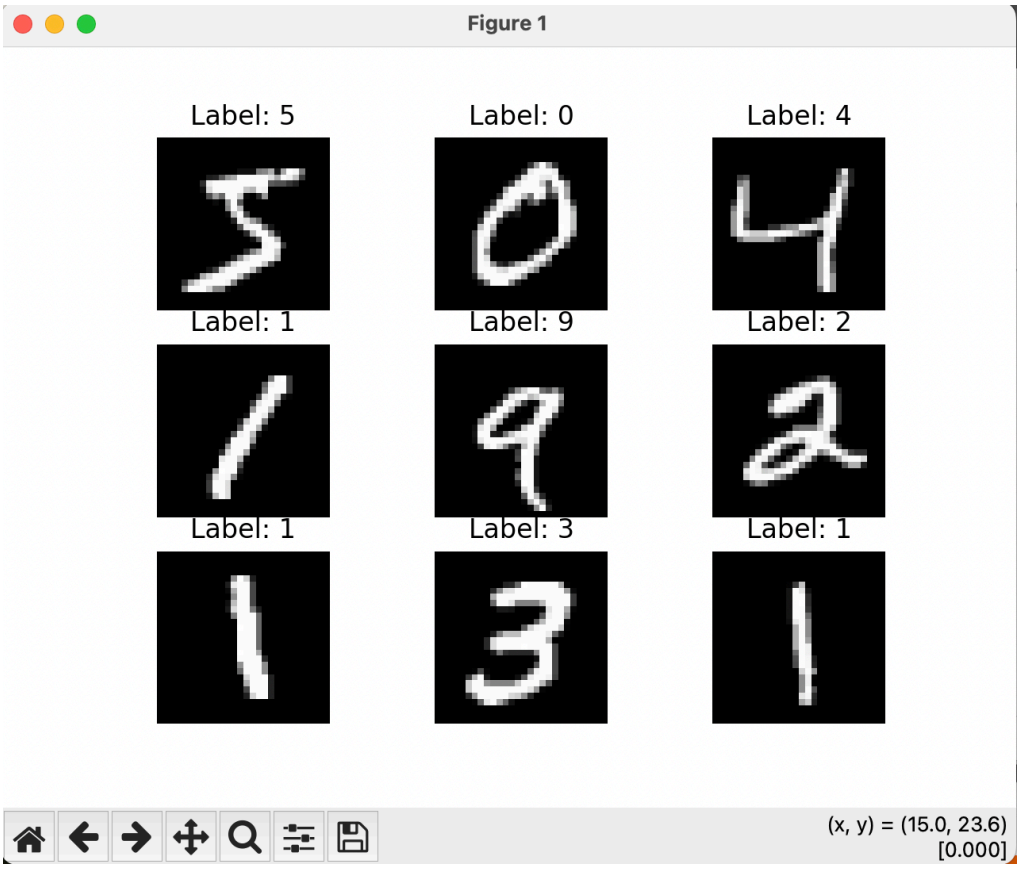
a. MNIST Dataset Preprocessing:

The MNIST dataset consists of grayscale images of handwritten digits (0-9). Preprocessing steps:

- **Loading the Dataset:** The MNIST dataset was loaded using the `tf.keras.datasets.mnist.load_data()` method, which splits the dataset into training and test sets.
- **Normalization:** Pixel values in the images, which are initially in the range `[0, 255]`, were normalized by dividing them by `255.0`. This scaled the pixel values to the range `[0, 1]`, which helps the neural network converge faster and improve training stability.

- **Expanding Dimensions:** Since the images are grayscale (single channel), the shape of each image is `(28, 28)`. To match the input shape expected by the CNN (which expects a 4D input of `(batch_size, height, width, channels)`), an extra dimension was added to represent the channel. Thus, the shape became `(28, 28, 1)`.
- **Label Encoding (One-Hot Encoding):** The labels in the MNIST dataset are integers (0-9). For the `categorical_crossentropy` loss function, these integer labels were one-hot encoded into a vector of length 10 (the number of classes). Each class is represented as a binary vector where the index corresponding to the label is 1, and all other values are 0.

Visualizing some MNIST Images:

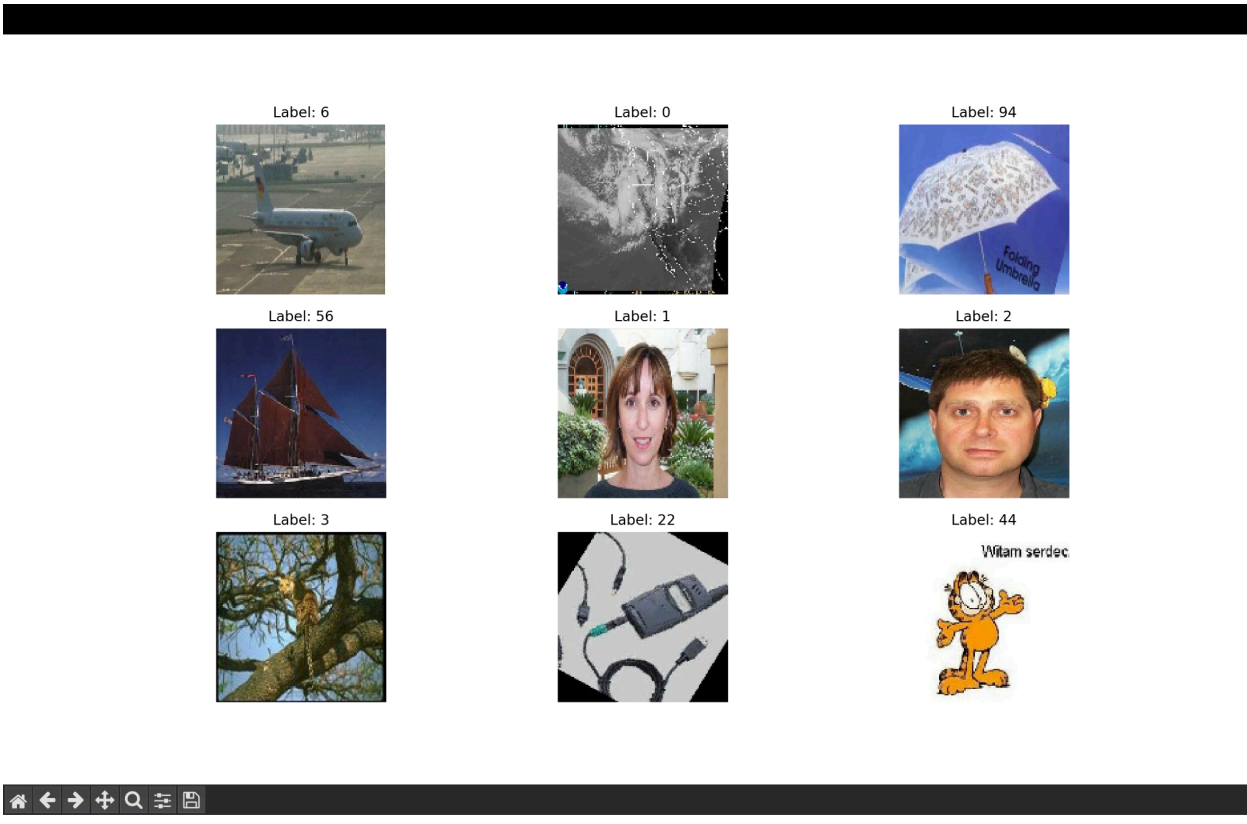


b. Caltech-101 Dataset Preprocessing:

The Caltech-101 dataset consists of 101 object categories with color images of varying sizes. The preprocessing steps performed were:

- **Loading the Dataset:** The dataset was loaded from a local directory using `tf.keras.preprocessing.image_dataset_from_directory()`. The function automatically loads images and labels from subdirectories and splits them into batches.
- **Resizing Images:** Since the images in the Caltech-101 dataset come in various sizes, they were resized to a consistent shape of `128x128` to ensure compatibility with the CNN. This is essential because the CNN requires inputs of fixed dimensions.
- **Splitting the Dataset:** The dataset was split into training and testing sets with an 80/20 ratio using TensorFlow's `take` and `skip` functions.
- **Label Encoding (One-Hot Encoding):** Similar to the MNIST dataset, the labels for the Caltech-101 dataset were also one-hot encoded. Since there are 101 classes in this dataset, the labels were encoded into a vector of length 101 for the `categorical_crossentropy` loss function.

Visualizing some Caltech Images:



3. CNN model Architecture

The Convolutional Neural Network (CNN) architecture used in the code is designed to process both the MNIST and Caltech-101 datasets. Below is a breakdown of the CNN model's architecture and the key components involved:

a. **Input Layer:**
   - The input layer is defined based on the dataset being used.
     - **For MNIST:** The input shape is `(28, 28, 1)` since the images are grayscale (single channel) and have a resolution of `28x28` pixels.
     - **For Caltech-101:** The input shape is `(128, 128, 3)` as the images are colored (3 channels) and resized to `128x128` pixels.

b. **First Convolutional Layer:**
   - The first convolutional layer has 32 filters with a kernel size of `(3x3)`. This layer applies a set of 32 filters to the input image to extract low-level features like edges, corners, and textures.
   - The activation function used is **ReLU** (Rectified Linear Unit), which introduces non-linearity into the model, helping it learn complex patterns.

c. **First Pooling Layer:**
   - A **MaxPooling2D** operation with a pool size of `(2x2)` is applied to reduce the spatial dimensions of the feature maps by half. This helps in reducing computational complexity and preventing overfitting by retaining only the most important features.

d. **Second Convolutional Layer:**
   - The second convolutional layer has 64 filters with a kernel size of `(3x3)`. This layer applies 64 filters to the output of the first pooling layer, allowing the network to learn more abstract and complex features.
   - The **ReLU** activation function is applied again to introduce non-linearity.

e. **Second Pooling Layer:**
   - The second pooling layer can be either **MaxPooling** or **AveragePooling**, depending on the configuration.
     - **MaxPooling:** Selects the maximum value in each pooling window.
     - **AveragePooling:** Takes the average value in each pooling window.

f. **Flatten Layer:**
   - After the second pooling layer, the 2D feature maps are **flattened** into a 1D vector, which is then passed into the fully connected layers.
   - Flattening converts the multidimensional output from the convolutional layers into a vector that can be processed by the dense layers.

g. **Fully Connected (Dense) Layer:**
   - The fully connected layer has 128 neurons and uses the **ReLU** activation function. This layer allows the network to combine the features extracted by the convolutional layers and learn complex relationships between them.

h. **Output Layer:**
   - The output layer has a number of neurons equal to the number of classes in the dataset:
     - **For MNIST:** There are 10 output neurons (one for each digit class 0-9).
     - **For Caltech-101:** There are 101 output neurons (one for each class in the dataset).
   - The **softmax** activation function is applied to ensure that the output values represent probabilities, where the sum of the probabilities is 1.

i. **Compilation:**
   - The model is compiled using the following options:
     - **Optimizer:** The code supports different optimizers such as `Adam`, `SGD`, and `RMSprop`, depending on the configuration.
     - **Loss Function:** The loss function can be either `categorical_crossentropy` (for one-hot encoded labels) or `sparse_categorical_crossentropy` (for integer-encoded labels).
     - **Metrics:** The model is evaluated using accuracy and Top-5 accuracy metrics.

4. Training and Evaluation

a. Training the CNN on MNIST and Caltech-101 Datasets

- MNIST Dataset:
  - Max Pooling:
    - Top 1 Accuracy: 99.11%
    - Top 5 Accuracy: 99.99%
  - Average Pooling:
    - Top 1 Accuracy: 99.13%
    - Top 5 Accuracy: 99.99%

  Both pooling methods performed almost equally well, with high accuracy on MNIST.

- Caltech-101 Dataset:
  - Max Pooling:
    - Top 1 Accuracy: 45.12%

        - Top 5 Accuracy: 60.34%
    - Average Pooling:
        - Top 1 Accuracy: 44.95%
        - Top 5 Accuracy: 61.61%

While Top 5 accuracy was decent, the Top 1 accuracy shows the challenge of classifying 101 classes.

b. Exploring Accuracy with Different Loss Functions and Optimizers:

i. Categorical Cross-Entropy with Adam:

MNIST: Excellent results, Top 1: 99.11%, Top 5: 100%.

Test accuracy (Top 1): 0.991599977016449
Test accuracy (Top 5): 1.0

Caltech-101: Top 1: 45.12%, Top 5: 60.34%.

Caltech-101 Test accuracy (Top 1): 0.4511858820915222
Caltech-101 Test accuracy (Top 5): 0.6039713025093079

ii. Categorical Cross-Entropy with SGD:

MNIST: Top 1: 96.60%, Top 5: 99.97%.

Test accuracy (Top 1): 0.9711999893188477
Test accuracy (Top 5): 0.9997000098228455

Caltech-101: Encountered instability, Top 1: 4.74%.

Caltech-101 Test accuracy (Top 1): 0.04743519052863121
Caltech-101 Test accuracy (Top 5): 0.0

iii. Sparse Categorical Cross-Entropy with Adam:

MNIST: Top 1: 99.18%, Top 5: 100%.

Test accuracy (Top 1): 0.9918000102043152
Test accuracy (Top 5): 0.9998999834060669

Caltech-101: Top 1: 44.12%, Top 5: 57.19%.

Caltech-101 Test accuracy (Top 1): 0.4412575960159302
Caltech-101 Test accuracy (Top 5): 0.5719801187515259

iv. Sparse Categorical Cross-Entropy with RMSprop:

MNIST: Top 1: 98.96%, Top 5: 100%.

Test accuracy (Top 1): 0.9896000027656555
Test accuracy (Top 5): 1.0

Caltech-101: Top 1: 43.96%, Top 5: 58.63%.

Caltech-101 Test accuracy (Top 1): 0.43960288166999817
Caltech-101 Test accuracy (Top 5): 0.5863209962844849

c. Conclusion:

- For MNIST, both pooling strategies (max and average) and loss function/optimizer combinations yielded high performance, with **categorical cross-entropy** and **Adam** being particularly effective.
- On the more complex Caltech-101 dataset, **sparse categorical cross-entropy** with **Adam** or **RMSprop** optimizers produced better results in terms of accuracy.
- **Top 5 accuracy** was consistently high across all configurations for both datasets, but the **Top 1 accuracy** on Caltech-101 showed room for improvement due to the dataset's complexity and the larger number of classes.

5. Pooling Variations

a. Task Overview:

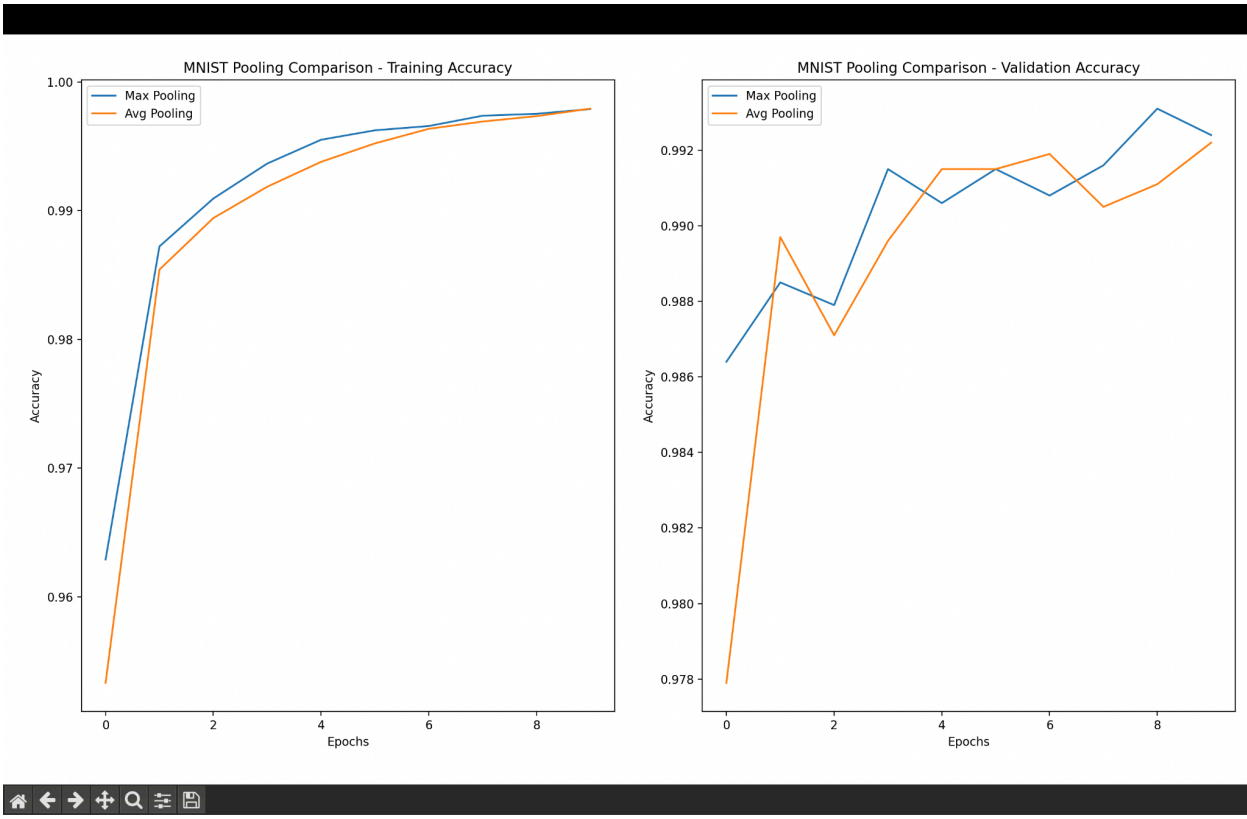
We experimented with two different pooling strategies for the second pooling layer in the CNN:

- Option A: Max Pooling (2x2 window)
- Option B: Average Pooling (2x2 window)

The CNN model was trained and evaluated on both the MNIST and Caltech-101 datasets using these pooling strategies, and the training accuracy, validation accuracy, and loss curves were compared.

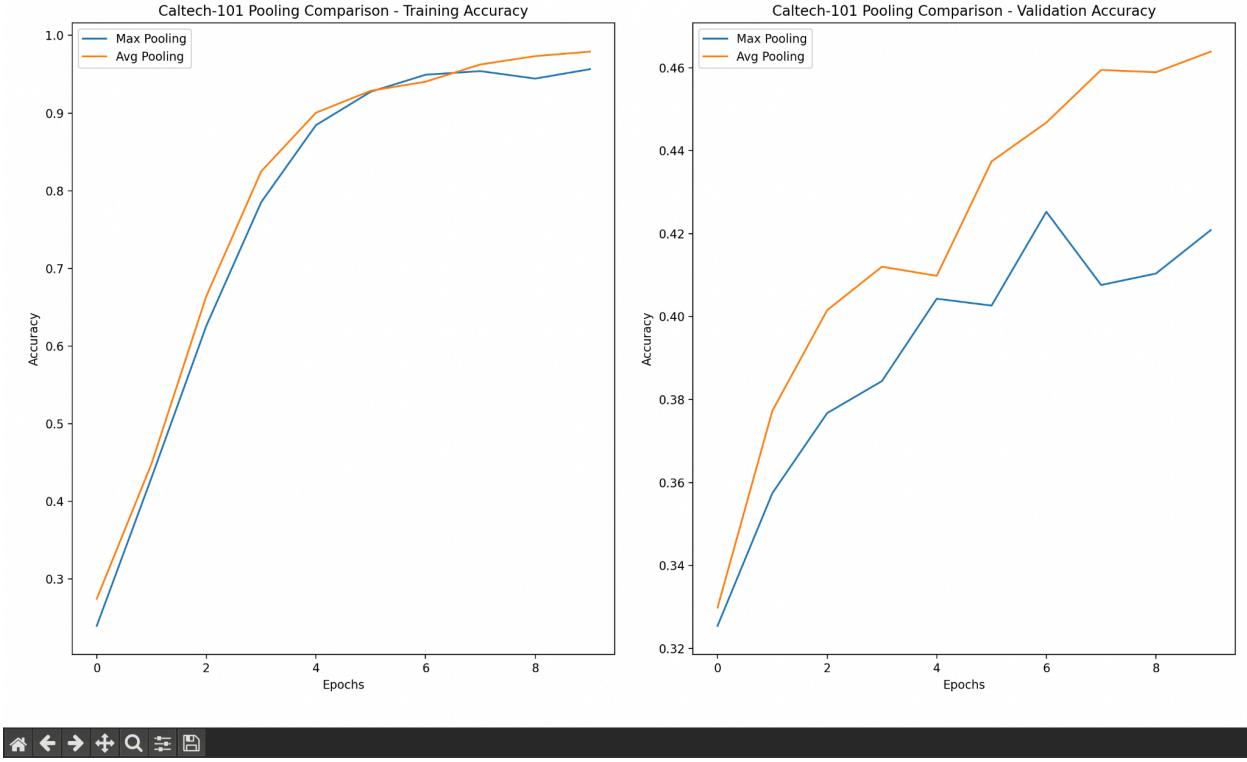
b. Results Comparison:

MNIST Dataset:



- Max Pooling:
  - Training Accuracy: 99.55%
  - Validation Accuracy: 99.11%
  - Test Loss: 0.0273
  - Top 1 Accuracy: 99.11%
  - Top 5 Accuracy: 99.99%
  - Convergence Speed: Faster convergence and marginally higher accuracy than Average Pooling.
- Average Pooling:
  - Training Accuracy: 99.41%
  - Validation Accuracy: 99.13%
  - Test Loss: 0.0278
  - Top 1 Accuracy: 99.13%
  - Top 5 Accuracy: 99.99%
  - Convergence Speed: Slower convergence but with a similar final accuracy to Max Pooling.

Caltech-101 Dataset:



- Max Pooling:
  - Training Accuracy: 92.44%
  - Validation Accuracy: 45.12%
  - Test Loss: 4.47
  - Top 1 Accuracy: 45.12%
  - Top 5 Accuracy: 60.34%
  - Convergence Speed: Max Pooling led to faster convergence, but both methods struggled with this complex dataset.
- Average Pooling:
  - Training Accuracy: 90.96%
  - Validation Accuracy: 44.95%
  - Test Loss: 4.42
  - Top 1 Accuracy: 44.95%
  - Top 5 Accuracy: 61.61%
  - Convergence Speed: Slower convergence compared to Max Pooling, but it marginally improved Top 5 accuracy.

c. **Impact of Pooling Strategies:**
    - Training Accuracy and Loss Curves:
        - On both datasets, Max Pooling provided faster convergence during training and performed slightly better in terms of training accuracy.
        - Average Pooling, on the other hand, took longer to converge, but still resulted in similar accuracy on MNIST and Caltech-101.
        - On Caltech-101, both pooling methods struggled due to the complexity of the dataset and the high number of classes.
    - Validation and Test Accuracy:
        - The MNIST dataset showed that both pooling methods resulted in near-perfect accuracies (99%+), with Max Pooling slightly outperforming in validation and test accuracy.
        - On the Caltech-101 dataset, Max Pooling led to better Top 1 accuracy, while Average Pooling performed slightly better in terms of Top 5 accuracy, indicating that it was better at recognizing similar classes.
d. **Conclusion**:
    - Max Pooling is generally better for faster convergence and is more efficient at retaining important features, which benefits models that require high performance in a short amount of time.
    - Average Pooling, while slower in convergence, smooths out the feature maps and can be beneficial in tasks that require a more generalized feature extraction. In the case of Caltech-101, it marginally improved Top 5 accuracy, likely because it was better at handling the variability between similar classes.

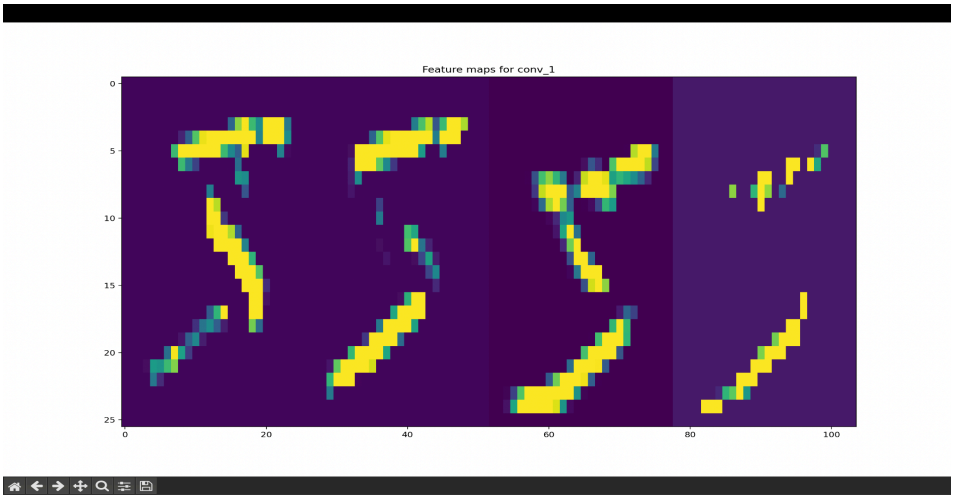
Overall, Max Pooling was slightly more efficient on both MNIST and Caltech-101 in terms of speed and accuracy, while Average Pooling showed some advantages in handling complex datasets with numerous classes.

6. Feature Map Visualization

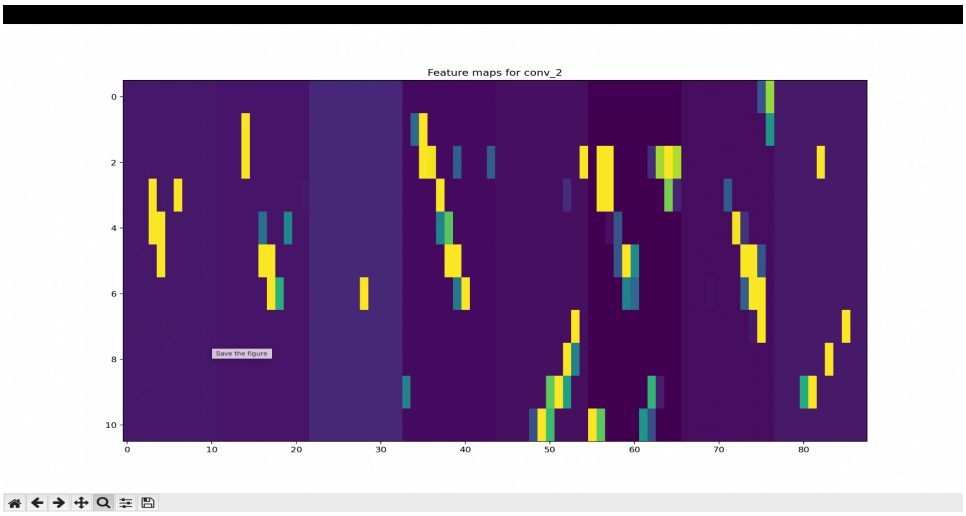
Feature maps from the first and second convolutional layers were visualized for one sample image from each class in both the MNIST and Caltech-101 datasets.

a. MNIST Feature Maps:

i. Conv Layer 1: The feature maps from the first convolutional layer showed the basic edges and textures in the digit images, highlighting sharp features.

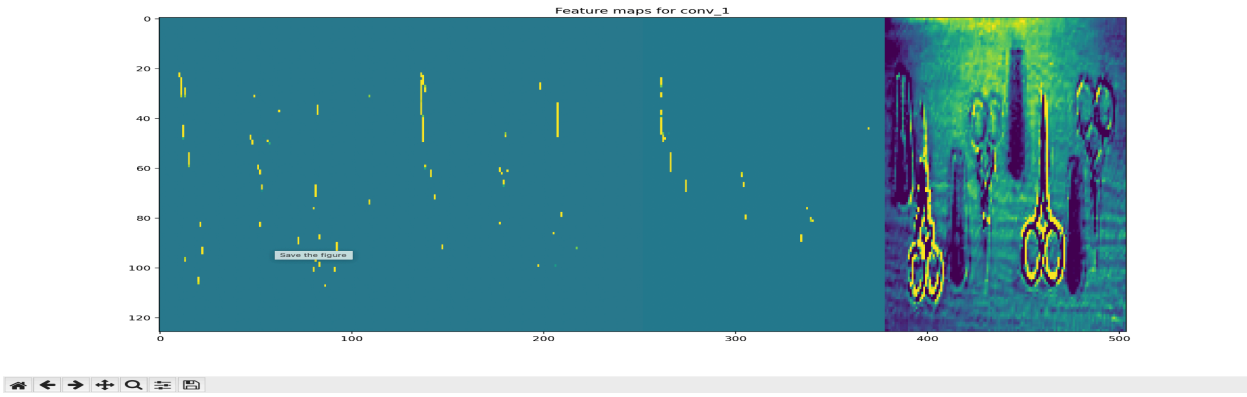


ii. Conv Layer 2: As we moved to deeper layers, the feature maps became more abstract, focusing on more complex patterns such as curves and combinations of edges.

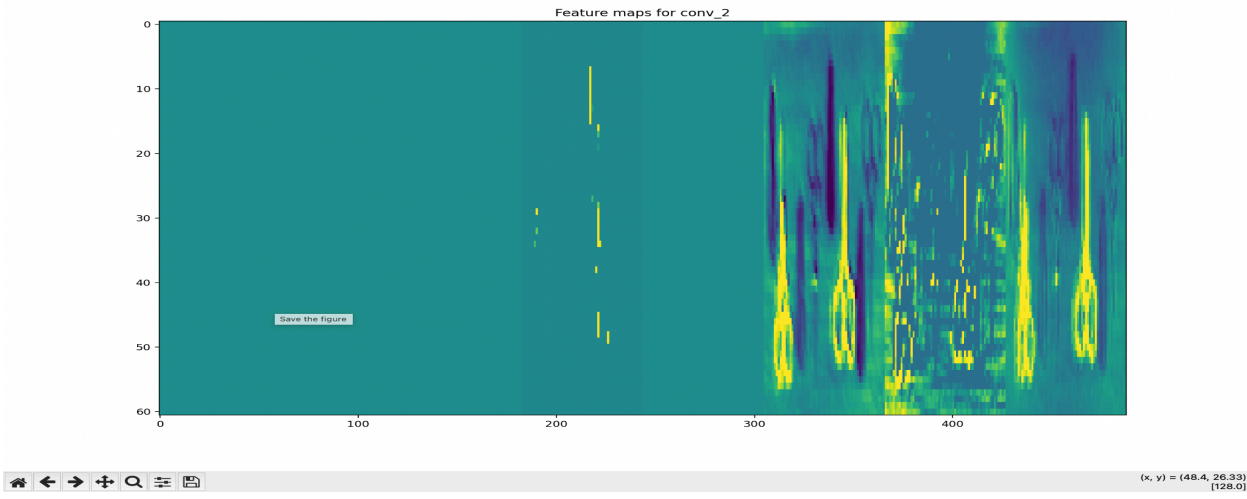


b. Caltech-101 Feature Maps:

i. Conv Layer 1: In the first layer, the feature maps primarily focus on basic texture elements of the images, capturing edges, corners, and textures of different classes.



ii. Conv Layer 2: Deeper layers were more specialized, focusing on larger, more complex features that represent the structure of the object, such as the overall shape of animals or faces.



c. **Analysis of Feature Maps**
    - First Convolutional Layer: Captures simple features like edges and corners. This is common for both datasets. For example, in MNIST, it identifies simple lines, while in Caltech-101, it captures textures and edges.
    - Second Convolutional Layer: These maps capture more complex patterns, such as the overall shape or larger structures. The network progressively focuses on global image features as depth increases.

d. **Conclusion on Feature Maps**
    - The feature maps evolve from simple edge detection to capturing more complex structures in both datasets.
    - The convolutional layers play a critical role in gradually building an understanding of the image from local features (edges, textures) to more complex features (shapes, objects).

7. **Conclusion:**
    - The CNN performed exceptionally well on MNIST, with high Top 1 and Top 5 accuracy across all configurations.
    - Caltech-101 posed more of a challenge due to its complexity, resulting in lower Top 1 accuracy but reasonable Top 5 accuracy.
    - Feature maps evolved from detecting basic patterns in the early layers to more complex features in the deeper layers.
    - **Adam** optimizer and **categorical cross-entropy** loss proved to be the most effective combination for both datasets. Pooling methods (Max and Average) showed similar performance but with slight differences in convergence speed.

- End -