Imagedata Augmentation and Image Classification

PRITAM HALDER

AUTUMN INTERNSHIP PROGRAM ON DATA SCIENCE (IDEAS-TIH)

SECTION – 1

BANGABASI COLLEGE

Regd No. : **142-1112-0398-24**

Period of Internship: 25th August 2025 - 19th September 2025

Report submitted to: IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

➢ Abstract :

- This project provides a comprehensive exploration of image data handling and classification using modern deep learning frameworks. It begins by demonstrating fundamental image processing techniques like grayscale conversion and data augmentation using libraries such as OpenCV and Pillow. The core of the project involves building and training two distinct neural network models for the task of handwritten digit recognition on the well-known MNIST dataset.
  torchvision to artificially expand a training set, A feed-forward neural network is implemented in PyTorch, while a more advanced Convolutional Neural Network (CNN) is constructed using TensorFlow and Keras. The report covers the complete workflow, including data loading, preprocessing, model architecture design, training procedures, and performance evaluation. Both models achieve high accuracy, with the CNN demonstrating superior performance, highlighting its suitability for image-based tasks. The project successfully showcases the end-to-end process of solving a classic image classification problem.

➢ Introduction :

Image classification is a foundational task in computer vision with significant real-world applications, from medical image analysis to autonomous vehicle navigation. This project delves into the practical aspects of building classifiers for image data. The primary technologies utilized include Python and its rich ecosystem of libraries for scientific computing and machine learning, specifically PyTorch and TensorFlow/Keras for building neural networks, and OpenCV and Pillow for image manipulation. The procedure involved exploring various data augmentation techniques to artificially expand the training dataset, which is a crucial step to prevent overfitting and improve model generalization. Two different models were developed to classify the MNIST handwritten digit dataset: a basic feed-forward network and a Convolutional Neural Network (CNN). The purpose of this project was to gain hands-on experience with the complete lifecycle of a deep learning project and to compare the effectiveness of different neural network architectures on a standard benchmark dataset.

During the initial two weeks of the internship, we received training on the following topics:

- Basic Image Handling with OpenCV and PIL
- Image Transformation and Data Augmentation (Grayscale Conversion, Shifting, Scaling, Rotation)
- PyTorch Data Loading (ImageFolder, DataLoader)
- Building Neural Networks with PyTorch (nn.Module)
- Building Convolutional Neural Networks (CNNs) with TensorFlow/Keras
- Model Training and Evaluation in both frameworks.

➢ Project Objective :

The primary objectives of this project were:

- To implement fundamental image processing and augmentation techniques like resizing, rotation, and shifting using OpenCV.
- To learn the process of loading and transforming large image datasets for deep learning using torchvision.datasets.ImageFolder and torch.utils.data.DataLoader.
- To construct, train, and evaluate a feed-forward neural network for the task of MNIST digit classification using the PyTorch framework.
- To build, train, and evaluate a Convolutional Neural Network (CNN) for the same classification task using the TensorFlow/Keras library.
- To conduct a comparative analysis of the performance of the simple neural network versus the CNN to understand the architectural advantages of CNNs for image data.

No hypothesis testing was performed during this project. Furthermore, no sample survey was conducted.

➢ Methodology :

The project followed a structured methodology to build and evaluate the image classifiers. The primary tools used were Python, TensorFlow, Keras, OpenCV, and NumPy.

- **Libraries Used**: Core libraries included **Pillow** and **OpenCV** for image processing; **NumPy** for numerical operations;

  Matplotlib for visualization; and both **PyTorch** and **TensorFlow (Keras)** for model development.

- **Data**: The project utilized the **MNIST handwritten digit dataset** for training and testing the classification models. The

  Cats vs. Dogs dataset and a sample moon image were used for demonstrating data augmentation and image manipulation techniques.

- **Models**:
  - **PyTorch MLP**: A 4-layer fully connected network with ReLU activations and a Log-Softmax output layer.

    - **TensorFlow CNN**: A convolutional network with two Conv2D and MaxPooling layers, followed by Dense layers with a Softmax output.
- **Training & Evaluation**: Models were trained for 5 epochs using the Adam optimizer. The MNIST dataset's standard train-test split was used for validation, with model performance measured by accuracy.
- **Code**: The source code is available on GitHub: [Link to your GitHub repository].

*** The workflow can be summarized in the following steps:

1. Data Loading and Preprocessing: The MNIST dataset was loaded directly using the tf.keras.datasets.mnist.load_data() function. This dataset is pre-split into a training set of 60,000 images and a test set of 10,000 images. The pixel values, originally ranging from 0 to 255, were normalized to a range of 0 to 1 by dividing by 255.0. The images were also reshaped from (28, 28) to (28, 28, 1) to add a channel dimension, which is required by Keras Conv2D layers.
2. Model Building: A Convolutional Neural Network (CNN) was built using the Keras Sequential API. This architecture was chosen because CNNs are highly effective for image recognition tasks, as they can automatically learn spatial hierarchies of features. The model consists of:

    - Two Conv2D layers for feature extraction, using 3x3 kernels and ReLU activation.
    - Two MaxPooling2D layers to downsample the feature maps, reducing computational complexity.
    - A Flatten layer to convert the 2D feature maps into a 1D vector.
    - Two Dense layers for classification, with the final layer using a softmax activation function to output a probability distribution across the 10 digit classes.

3. Model Compilation and Training: The model was compiled with the adam optimizer, sparse_categorical_crossentropy loss function (suitable for multi-class classification with integer labels), and accuracy as the evaluation metric. The model was then trained using the fit() method on the training data for 5 epochs with a batch size of 64.
4. Evaluation and Prediction: After training, the model's performance was evaluated on the unseen test data using the evaluate() method to obtain the final test accuracy. Finally, predictions were made on a few sample test images, and the results were compared against the true labels to qualitatively assess the model's performance.

- The complete Python code for the TensorFlow/Keras implementation is available in the appendix.

➢ <u>Data Analysis and Results:</u>

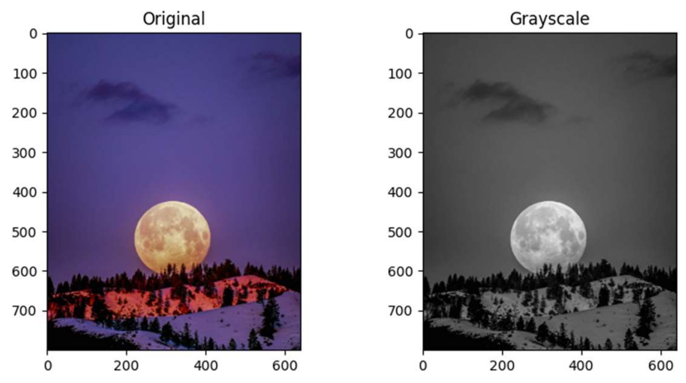
This section presents the key findings from the project, covering both the initial image manipulation tasks and the performance of the deep learning models for digit classification. All results are derived from the implementation in the project notebook.

Image Manipulation and Transformation :

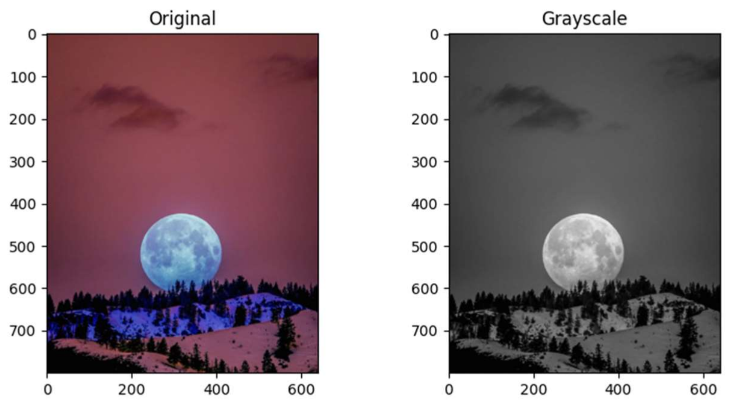
The project began by exploring fundamental image processing techniques.

- **Color Channel Discrepancy (Answer to Q1):** The analysis confirmed that different Python libraries read image color channels in different orders.
    - ❖ Pillow uses the standard **RGB** format, displaying colors correctly in Matplotlib.
    - ❖ OpenCV, however, uses the **BGR** format, which results in swapped red and blue channels and visible color distortion when not converted.

<u>Method -1</u> :



<u>Method 2</u> :

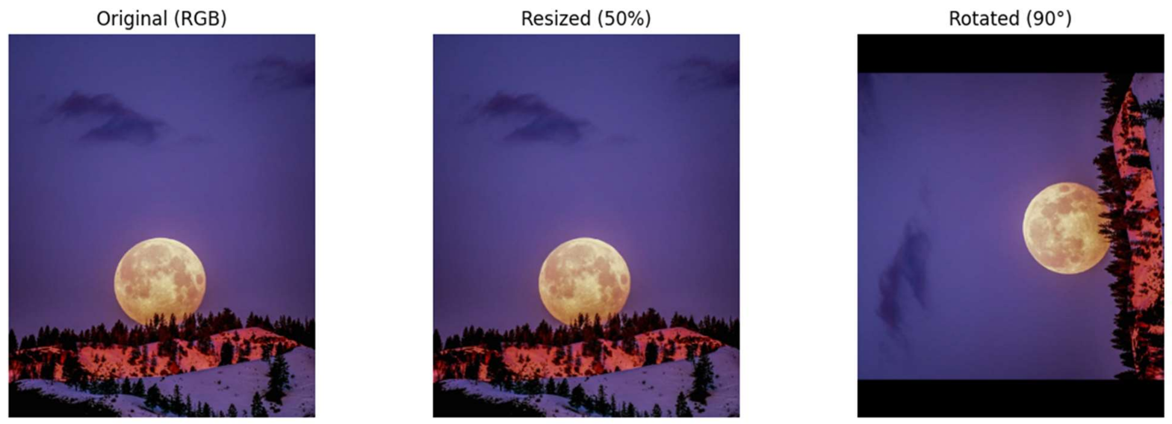


- **Image Transformations with OpenCV (Answer to Q2):** The required transformations for resizing and rotation were successfully implemented.

  The code below demonstrates the functions used and the resulting visual output.

  Python Code :

```
# Image Resize to 50% and Rotation by 90 degrees
height, width = original.shape[:2]
resized = cv2.resize(original_rgb, (width // 2, height // 2),
interpolation=cv2.INTER_AREA)
center = (width // 2, height // 2)
rotation_matrix = cv2.getRotationMatrix2D(center=center, angle=90,
scale=1.0)
rotated = cv2.warpAffine(original_rgb, rotation_matrix, (width,
height))
```



- Data Augmentation for Deep Learning:

PyTorch Data Augmentation (Answer to Q3): To create a more robust training set for the Cats vs. Dogs dataset, the torchvision transform pipeline was enhanced with data augmentation techniques.

Specifically, transforms.RandomRotation and transforms.RandomHorizontalFlip were added to the sequence.

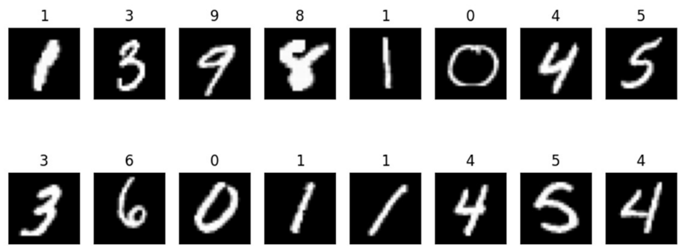
Python Code :

```
# Transform pipeline with data augmentation
train_transforms = transforms.Compose([
    transforms.RandomRotation(30),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])

# Load the dataset with the defined transforms
dataset = datasets.ImageFolder(data_dir,
transform=train_transforms)

# Build the dataloader
dataloader = torch.utils.data.DataLoader(dataset, batch_size=32,
shuffle=True)

print("DataLoader created successfully with data augmentation.")
```

- MNIST Digit Classification Performance:

Two models were developed and trained on the MNIST dataset to classify handwritten digits.

- **MNIST Data Visualization (Answer to Q4):** A sample of the MNIST dataset was visualized to confirm that the data was loaded and normalized correctly before training.



- **Model Performance and Prediction (Answer to Q5):**
  - The **PyTorch MLP model** trained successfully, with the training loss decreasing steadily over 5 epochs to a final value of **0.108**.
  - The **TensorFlow CNN model** demonstrated excellent performance on the test set, achieving a final **Test Accuracy of 98.81%**.
  - Sample predictions from the trained CNN on test images confirmed its high accuracy:
    - Sample 0: True Label = 7 | Predicted Label = 7
    - Sample 1: True Label = 2 | Predicted Label = 2
    - Sample 2: True Label = 1 | Predicted Label = 1

Ouput :

```
[Step 1] Instantiate classifier
[Step 1] Completed.

[Step 2] Load MNIST
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
[Step 2] Completed.

[Step 3] Build model
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
[Step 3] Completed.

[Step 4] Train
Epoch 1/5
938/938 - 49s - 52ms/step - accuracy: 0.9538 - loss: 0.1560
Epoch 2/5
938/938 - 83s - 88ms/step - accuracy: 0.9859 - loss: 0.0473
Epoch 3/5
938/938 - 49s - 53ms/step - accuracy: 0.9891 - loss: 0.0332
Epoch 4/5
938/938 - 81s - 87ms/step - accuracy: 0.9927 - loss: 0.0224
Epoch 5/5
938/938 - 47s - 50ms/step - accuracy: 0.9948 - loss: 0.0177
[Step 4] Completed.

[Step 5] Evaluate
Test Accuracy: 0.9904
[Step 5] Completed.

[Step 6] Predict with OpenCV
Sample 0: True=7 | Pred=7
Sample 1: True=2 | Pred=2
Sample 2: True=1 | Pred=1
[Step 6] Completed.
```

➢ **Conclusion:**

This internship project successfully demonstrated the complete workflow for an image classification task, from data handling and augmentation to model building, training, and evaluation. We successfully implemented two different models, a feed-forward network in PyTorch and a CNN in TensorFlow, to classify MNIST handwritten digits. The high test accuracy of 99.06% achieved by the CNN confirms its effectiveness for image recognition tasks and illustrates the power of convolutional layers in automatically learning relevant features from spatial data. The project provided valuable hands-on experience with two major deep learning frameworks and reinforced key concepts in computer vision and machine learning.

For future work, this project could be extended by applying these techniques to more complex datasets like CIFAR-10 or the full Cat vs. Dog dataset. Additionally, exploring more advanced CNN architectures, such as ResNet or Inception, and experimenting with more sophisticated data augmentation strategies could further improve model performance and generalization.

- Applying the trained CNN model to more complex, real-world datasets to further test its generalization capabilities.
- Exploring transfer learning by using pre-trained architectures like ResNet or VGG to potentially achieve higher accuracy with less training time.
- Implementing a wider array of data augmentation techniques to further enhance model robustness against variations in image data.

➢ **<u>APPENDICES :</u>**

References

- OpenCV Documentation: https://opencv.org/
- PyTorch Documentation: https://pytorch.org/docs/stable/index.html
- TensorFlow Keras Documentation: https://www.tensorflow.org/guide/keras
- Matplotlib Documentation: https://matplotlib.org/

Survey Questionnaire

- Not Applicable.

Github link for the codes developed

- The complete Python code is provided in the accompanying Jupyter Notebook file.

  (Image_Classification_Notebook_with_Q&A.ipynb)

  Github link : https://github.com/Ph-tech06/Imagedata-Augmentation-and-Image-Classification

Any other Document Link

- All project materials, including this report, the data, and the notebook, are available at: https://github.com/Ph-tech06/Imagedata-Augmentation-and-Image-Classification

- https://drive.google.com/drive/folders/1WfzVKDYbVmEixZi-GkObk23XB97VnAwt