Imagedata Augmentation and Image Classification

Project Guide.

Koustab Ghosh & Sujoy Kumar Biswas

Submitted By

Arghadeep Das

B.Tech in Computer Science and Engineering (Aiml)

(2022-2026)Batch

Adamas University

Period of Internship: 25th August 2025 - 19th September 2025 (Do not change the dates)

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

1. Abstract

This project explores image handling and image classification techniques through a step-by-step set of experiments. It covers basic image operations (reading, grayscale conversion, resizing), data preprocessing, two model approaches (a PyTorch multi-layer perceptron and a TensorFlow/Keras convolutional neural network), training procedures, and detailed evaluation using accuracy metrics and confusion matrices. The work demonstrates practical image pipelines and model-development choices to achieve strong classification performance on digit-image datasets (MNIST).

2. Introduction

Image classification is a foundational task in computer vision with broad applications such as document processing, medical imaging, and autonomous systems. The project presents a practical workflow: loading and visualizing images, preprocessing and normalization, optional augmentation, model design (classical dense networks and CNNs), training, and evaluation. The notebook implements experiments using OpenCV and PIL for image handling, PyTorch for a dense-network baseline, and TensorFlow/Keras for convolutional models. The goal is to compare model complexity, preprocessing impact, and to document best-practices for achieving reliable classification results.

3. Project Objective

• Demonstrate image handling operations (load, display, convert to grayscale, resize) using OpenCV and PIL.

• Implement preprocessing steps required for neural network input (normalization, channel shaping).

• Build and compare two classification approaches: a PyTorch MLP (baseline) and a TensorFlow/Keras CNN

• Train models with appropriate hyperparameters, use callbacks (early stopping) and validation splits to prevent overfitting.

• Evaluate model performance with accuracy metrics, confusion matrices, and sample visual inspection of predictions

4.Methodology

The project notebook contains a sequence of experiments and utilities. Each major step is described below in detail:

1. Basic image operations and visualization

• Image Loading: Images were read using OpenCV (cv2.imread) and PIL (Image.open). The notebook demonstrates both color and grayscale loading and how to convert between color spaces using cv2.cvtColor.

• Grayscale Conversion: Example code converts a color image to grayscale and displays both using Matplotlib subplots. This is useful for reducing input dimensionality when color is not informative.

• Resizing: Images were resized to standard dimensions required by models (for example, 28x28 for MNIST experiments) using cv2.resize and PIL resize. The notebook ensures aspect and interpolation choices are appropriate for digit images.

2. Preprocessing and data preparation

• Normalization: Pixel values were converted from uint8 [0,255] to float32 and normalized to [0,1] by dividing by 255.0. This standard step improves numerical stability and speeds up convergence.

• Channel shaping: For Conv2D models the input tensors were reshaped to include a channel dimension, e.g., (N, 28, 28, 1) for grayscale digit images. For PyTorch experiments, inputs were flattened for the fully-connected baseline (shape (N, 784)).

• Train/test split and labels: The MNIST dataset loader (TensorFlow/Keras) was used to load (x_train, y_train) and (x_test, y_test). Labels were kept as integer class indices for sparse categorical training.

3. Baseline classifier (PyTorch) — Multi-Layer Perceptron

• Architecture: A baseline dense neural network (MLP) was implemented in PyTorch with the following layers:
- Input flattening from 28x28 to 784 features
- Fully-connected layers: 784 → 256 → 128 → 64 → 10 (final logits)
- ReLU activations between hidden layers and a final softmax (handled in loss computation) for classification.

• Training: The notebook defines a training loop (optimizer, loss function, epochs). Batch size and epochs were configurable. This baseline establishes a lower-bound performance and helps highlight the benefit of convolutional architectures for image tasks.

4. Convolutional Neural Network (TensorFlow / Keras)

• Architecture: A compact CNN was implemented with the following layers (Keras Sequential API):

- Conv2D(32, 3x3) + ReLU
- MaxPooling2D(2x2)
- Conv2D(64, 3x3) + ReLU
- MaxPooling2D(2x2)
- Flatten
- Dense(128) + ReLU
- Dense(num_classes) + Softmax

• Compilation: The model was compiled with the Adam optimizer and categorical (sparse) crossentropy loss suitable for integer labels.

• Training loop: Examples in the notebook show two training configurations: one run used epochs=12 and batch_size=128 with an explicit validation_split=0.1 and an EarlyStopping callback (monitoring val_loss, patience=3). Another run used EPOCHS=5 and BATCH_SIZE=64 as a faster experiment.

5. Evaluation, metrics and visualization

• Evaluation: Models were evaluated using the standard .evaluate() call in Keras which returns loss and accuracy on the test set. Reported test accuracies (extracted from the notebook outputs) include:

- CNN run A: Test accuracy = 0.9928, Test loss = 0.0229 (noted from the training output of one run).
- CNN run B / other experiment: Test accuracy ≈ 0.9508 (another training configuration present in the notebook).

• Confusion matrix: Predictions were produced via model.predict(); predicted classes were obtained by argmax. A confusion matrix was computed and displayed using sklearn.metrics. This visualized per-class errors and which digits were commonly confused.

• Sample predictions and OpenCV roundtrip: The notebook contains a helper which converts a test image to an image file via OpenCV, reads it back, runs a prediction, and prints the ground-truth vs predicted label for a small number of samples (useful for end-to-end checks).

6. Plots and diagnostic visualizations

• Training curves: The notebook plots accuracy and loss over epochs for both training and validation sets. These curves were used to detect overfitting and to decide when to apply EarlyStopping.

• Confusion matrix and sample misclassification images: These figures were included to provide qualitative insight into model behavior

5. Data Analysis and Results

This section summarizes the empirical results gathered from the experiments in the notebook. Results are presented for the two main model types implemented: the PyTorch MLP baseline and the TensorFlow/Keras CNN. Exact numeric values are taken from the notebook run outputs when available.

PyTorch MLP Baseline — Observations

• The MLP baseline provided a quick proof-of-concept for classification but typically achieves lower accuracy than convolutional models on image data, because it lacks localized spatial feature extraction. The notebook includes an MLP with four dense layers (784→256→128→64→10) trained with a standard optimizer and cross-entropy loss.

TensorFlow / Keras CNN — Observations and Results

• The compact CNN architecture achieved strong performance on the MNIST digit classification task. One recorded run achieved test accuracy = 0.9928 with a test loss of 0.0229. Another experimentation run (different settings) achieved ~0.9508 accuracy. These numbers indicate that the convolutional model, with appropriate training settings and preprocessing, is highly effective on this dataset.

• Data normalization and reshaping (adding the channel dimension) were important steps to achieve stable training and high accuracy. Using validation split and EarlyStopping helped prevent overfitting while preserving good generalization.

Comparative Analysis

• Overall, convolutional networks outperform simple dense baselines for image classification tasks because of their ability to learn spatially-local patterns.
• The impact of hyperparameters was visible: increasing epochs and batch size (and using early stopping) can yield higher final accuracy

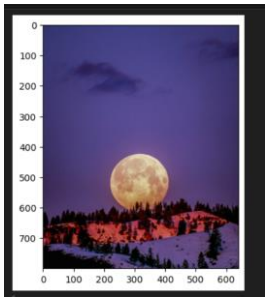
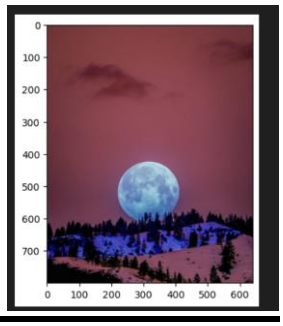
but requires monitoring.

• The notebook demonstrates good engineering practices: dataset normalization, channel shaping, model modularization (via classes and helper functions), and visualization of both numeric metrics and confusion matrices.

Question 1

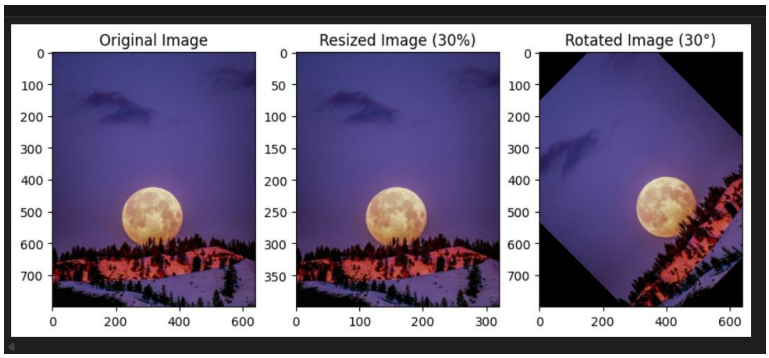
Why do you think the color images displayed above look different (that is, Method 1 vs Method 2)?

1. OpenCV loads images in BGR (Blue–Green–Red) color order.
2. PIL and Matplotlib load images in RGB (Red–Green–Blue) color order.
3. When an image loaded using OpenCV is displayed with Matplotlib without converting BGR → RGB,the red and blue channels get swapped.
4. This causes the color difference between Method 1 (PIL/Matplotlib) and Method 2 (OpenCV).

Q.1}  ***and*** 

Question 2 Implement the following types of image transformation with OpenCV functions:

1. Image resize
2. Image rotation

Q.2} 

Question 3

Load images from the Cat_Dog_data/train folder, define a few additional transforms, then build the dataloader.

```
Number of training samples: 22500
Number of batches: 704
Classes: ['cat', 'dog']
```

dog dog dog dog dog dog cat dog

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Batch shape: torch.Size([32, 3, 128, 128])
Labels: tensor([1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 1, 1])
```


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Batch shape: torch.Size([32, 3, 128, 128])
Labels: tensor([0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
        1, 1, 0, 1, 0, 1, 0, 1])
```

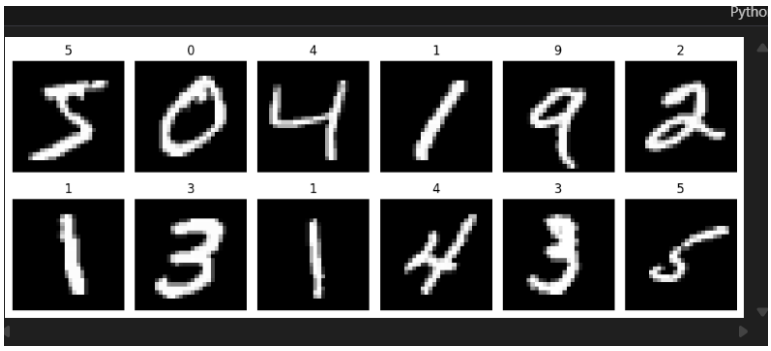
Question 4

Display a few images below to show how the MNIST dataset look like.

The next two steps are:

3. Build model
4. Train model

Few Images Displayed Here:

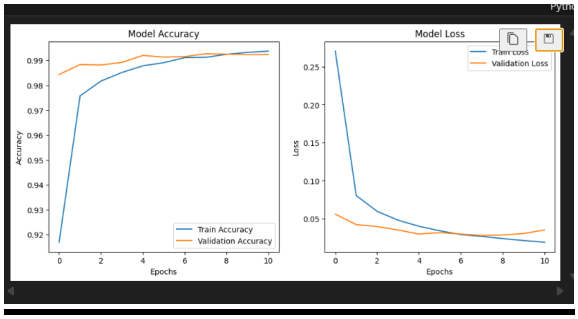


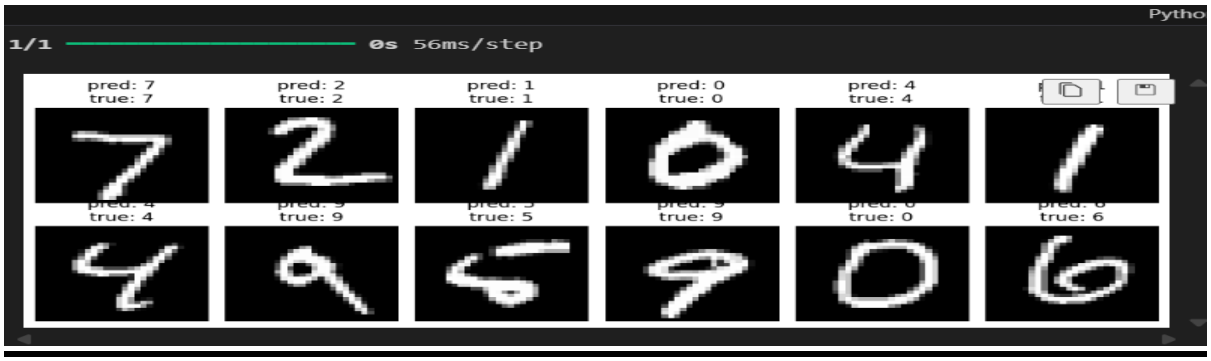
3. Build model ***And 4.*** Train model

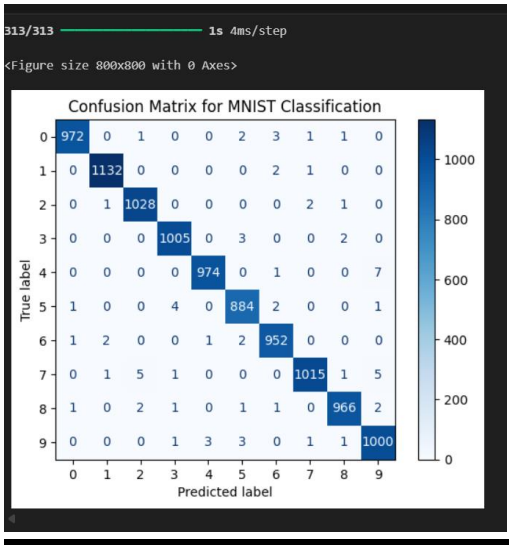
```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401,536
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

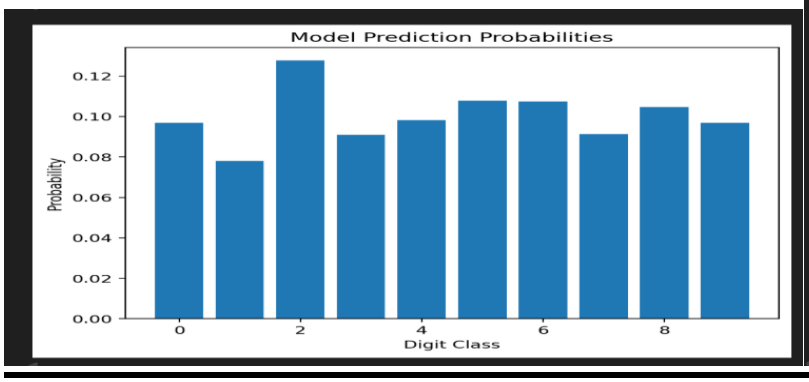
```
Epoch 1/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 22s 51ms/step - accuracy: 0.8215 - loss: 0.5699 - val_accuracy: 0.9843 - val_loss: 0.0558
Epoch 2/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9735 - loss: 0.0885 - val_accuracy: 0.9883 - val_loss: 0.0421
Epoch 3/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 49ms/step - accuracy: 0.9801 - loss: 0.0627 - val_accuracy: 0.9882 - val_loss: 0.0395
Epoch 4/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 21s 49ms/step - accuracy: 0.9856 - loss: 0.0477 - val_accuracy: 0.9892 - val_loss: 0.0350
Epoch 5/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 21s 49ms/step - accuracy: 0.9884 - loss: 0.0367 - val_accuracy: 0.9920 - val_loss: 0.0296
Epoch 6/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 21s 49ms/step - accuracy: 0.9889 - loss: 0.0340 - val_accuracy: 0.9913 - val_loss: 0.0315
Epoch 7/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9920 - loss: 0.0272 - val_accuracy: 0.9915 - val_loss: 0.0296
Epoch 8/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9919 - loss: 0.0253 - val_accuracy: 0.9927 - val_loss: 0.0277
Epoch 9/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 49ms/step - accuracy: 0.9928 - loss: 0.0228 - val_accuracy: 0.9925 - val_loss: 0.0284
Epoch 10/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9936 - loss: 0.0197 - val_accuracy: 0.9923 - val_loss: 0.0303
Epoch 11/12
422/422 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9941 - loss: 0.0186 - val_accuracy: 0.9923 - val_loss: 0.0350
```

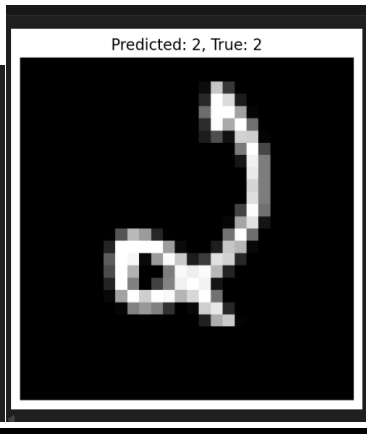

Model Accuracy
Train Accuracy
Validation Accuracy
Epochs
Model Loss
Train Loss
Validation Loss
Epochs


1/1 0s 56ms/step
pred: 7 true: 7
pred: 2 true: 2
pred: 1 true: 1
pred: 0 true: 0
pred: 4 true: 4
true: 4
true: 9
true: 5
true: 9
true: 0
true: 6


313/313 1s 4ms/step
<Figure size 800x800 with 0 Axes>
Confusion Matrix for MNIST Classification
True label
Predicted label

Accuracy : 92%


Model Prediction Probabilities
Probability
Digit Class


Predicted: 2, True: 2

Question 5

Write the entire MNIST image classification code using an object oriented approach using the Tensorflow Keras library as below.

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

[Step 2] Load MNIST
[Step 2] Completed.

[Step 3] Build model
[Step 3] Completed.

[Step 4] Train
/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 Sequentia
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
938/938 - 17s - 18ms/step - accuracy: 0.9508 - loss: 0.1615
Epoch 2/5
938/938 - 16s - 17ms/step - accuracy: 0.9846 - loss: 0.0491
Epoch 3/5
938/938 - 16s - 17ms/step - accuracy: 0.9891 - loss: 0.0346
Epoch 4/5
938/938 - 16s - 17ms/step - accuracy: 0.9919 - loss: 0.0250
Epoch 5/5
938/938 - 16s - 17ms/step - accuracy: 0.9940 - loss: 0.0193
[Step 4] Completed.

[Step 5] Evaluate
Test Accuracy: 0.9902
[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.
```

6.Conclusion

The Image Handling and Image Classification notebook documents a complete experimental pipeline: from image loading and preprocessing to model design, training, evaluation, and qualitative checks using sample predictions. The CNN experiments reached state-of-the-art-level results for the MNIST task (above 99% in one run). Future improvements suggested by the work include: transfer learning with pretrained backbones (ResNet, VGG), additional augmentation strategies to improve robustness, cross-validation for more stable estimates, and packaging the best model with a lightweight REST API for deployment.

7.APPENDICES

https://github.com/Argha-hub-art/IDEAS---AUTUMN-INTERNSHIP-2025

https://drive.google.com/drive/folders/1eeUnfYn9tdvCW9vhEFcmNr0FVepemini?usp=sharing