CIS 602 Fundamentals of Deep Learning
Fall 2025

Project 2
SmallCountNet CNN for Image Detection
Due: November 4, 2025

Michael Evan 02081213
Graduate Computer Science Department
UMassDartmouth

Abstract

Build a small convolutional neural network to count simple shapes: circles, squares, and triangles in 64×64 grayscale images. A dataset generator within JupyterNotebook places a random number (0-5) of each shape per image while preventing overlap, and filenames plus ground-truth counts are saved to CSVs. The CNN model (SmallCountNet) uses three conv -> ReLU -> max-pool blocks, adaptive average pooling, and a small fully connected head. Training uses the Adam optimizer with L1 loss (MAE) and an 80/10/10 train/validation/test split; the final bias is initialized to training label means to stabilize learning.

Results show the model predicts counts with very low error: per-class MAE typically -0.08 - 0.20 and RMSE similarly small, with most samples predicted within ±1 of the true count (roughly 85-100% depending on shape). Squares for some reason are the hardest class to predict. The pipeline includes learning curves, scatter and confusion matrices, and error distributions. Next steps are robustness tests (more overlap, varied sizes/noise) and evaluation on out-of-distribution images.

Introduction

1. Relevance & Importance

The relevant & importance of the data is that the simple shapes dataset teaches a real skill: counting objects in images. If a model can reliably count circles/squares/triangles it shows the pipeline (data, labels, CNN, metrics) works, and that same approach applies to real problems like counting cells, stars, or defects. The data is cheap & controlled so you can test ideas fast, find bugs, and improve robustness before moving to messy real world images.

2. Task Definition

Inputs: one grayscale image (IMG_SIZE × IMG_SIZE pixels): each pixel is a numeric brightness value.

Outputs: three labels - integer counts [#circles, #squares, #triangles] (e.g., 0 - 5).

Goal: given an image, predict those three counts as accurately as possible (training uses images + CSV label pairs).

3. Major contributions

1. Built a simple image generator that makes 64×64 grayscale pictures with random circles, squares, and triangles and saves the true counts to a CSV file.

2. Prevented shapes from overlapping so the labels match what actually appears in each image.

3. Designed and trained a small convolutional neural network (SmallCountNet) to predict three numbers: how many circles, squares, and triangles are in an image.

4. Set up training with a proper data split (train/validation/test), used the Adam optimizer and L1 (MAE) loss function, and saved the best model automatically.

5. Wrote reusable functions and notebook cells for dataset creation, training, evaluation, and plotting so the work is easy to rerun and display.

6. Measured performance with clear metrics (MAE, RMSE, % within ±k), and visual reports (learning curves, scatter plots, confusion matrices, error histograms and box plots).

7. Produced a final result summary and tools to inspect failures (worst images), so we can test robustness and improve the model.

Methodology

Data set

- Script written in JupyterNotebook creates images and a CSV (filenames + true counts). Shapes placed with no overlap. Stored in "shape_data" directory.

- Created images made: 64×64 grayscale images each of random number (0 - 5) circles, squares, triangles .

Data prep

- Load CSVs (train/val/test) and Images into a PyTorch Dataset that returns (image, label).
- Transforms: Resize(IMG_SIZE), ToTensor(), Normalize([0.5],[0.5]).
- Use DataLoader(batch_size = BATCH, shuffle) for training/validation/test.

Neural network architecture (see addendum for in-depth description)

- Simple conv -> pool -> dense CNN (SmallCountNet).
- Three blocks: Conv(1->16) -> pool, Conv(16->32) -> pool, Conv(32->64) -> pool.
- AdaptiveAvgPool -> Flatten -> Linear(64->64) ReLU -> Linear(64->3).
- Regression output (3 real numbers). No heavy regularization; final bias initialized to label means.

Training

- Split: 80% train / 10% val / 10% test.
- Optimizer: Adam, lr = 1e-3. Batch size = 64 (configurable). Epochs = configurable (e.g., 50-5000) only limited by time & computing resources.
- Loss: L1Loss (MAE). No per-class weighting (counts are regression targets).
- Save best model by lowest validation loss.

Figures for evaluation

- Architecture diagram (layer sizes).
- Learning curves (train vs val loss).
- Scatter plots actual vs predicted per class.
- Confusion matrices (rounded predictions).
- Error histogram & box plots,
- MAE, RMSE, GT/P
- Accuracy, Precision_Macro, Recall_Macro, F1_Macro

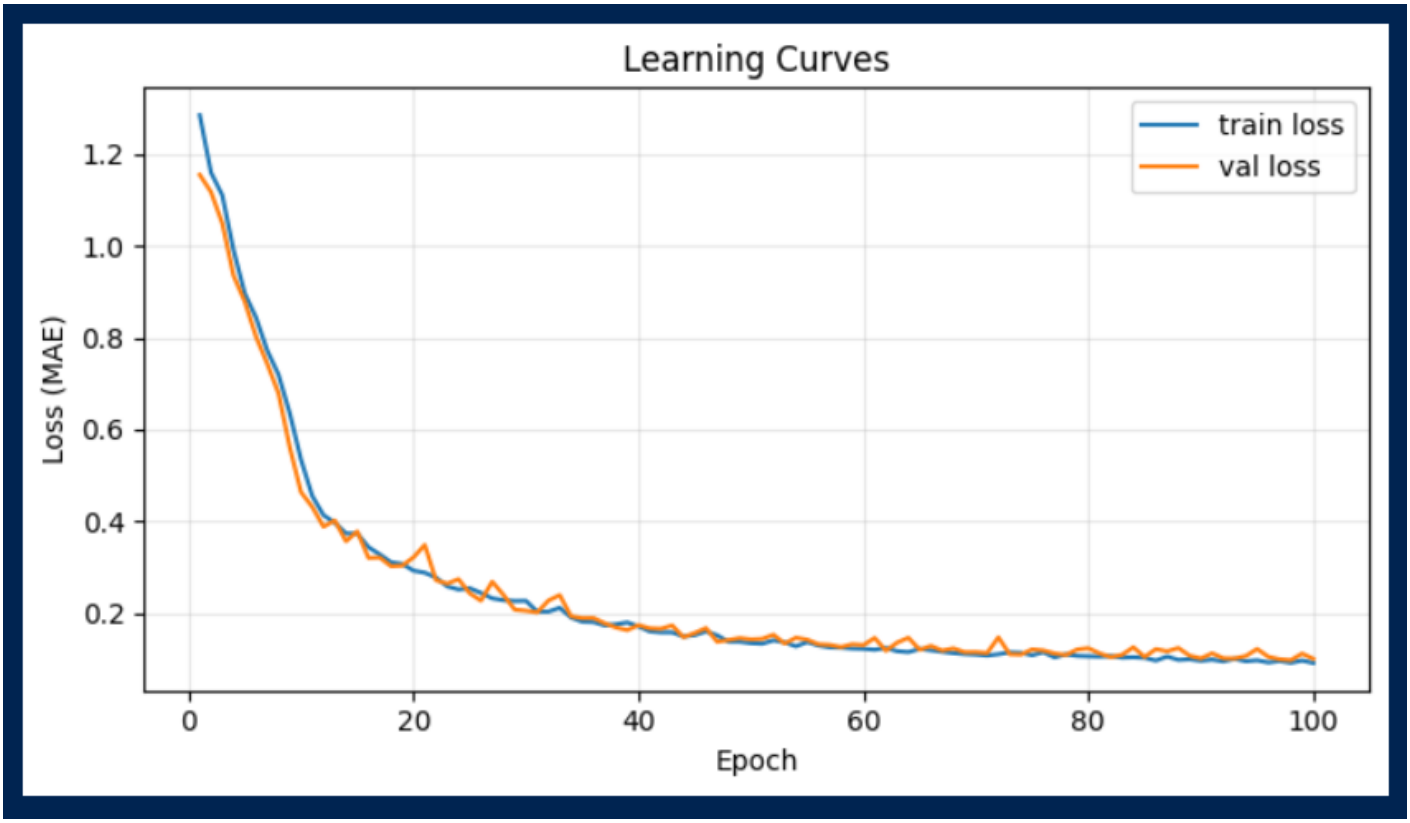
Results & Discussion

Note: results below are from 5000 64x64 images, batch = 64, & 100 epochs

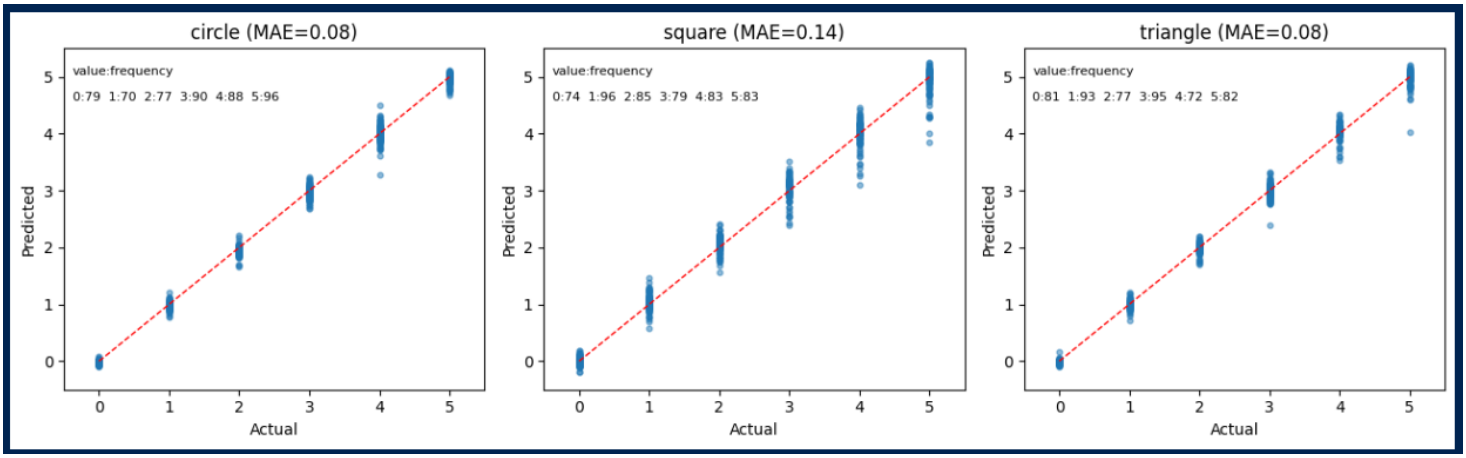
Classification metrics table on test set

class:	accuracy	precision_macro	recall_macro	f1_macro
circle:	0.998000	0.998168	0.998106	0.998127
square:	0.968000	0.968287	0.967566	0.967519
triangle:	0.996000	0.995580	0.996213	0.995871
macro_avg:	0.987333	0.987345	0.987295	0.987172

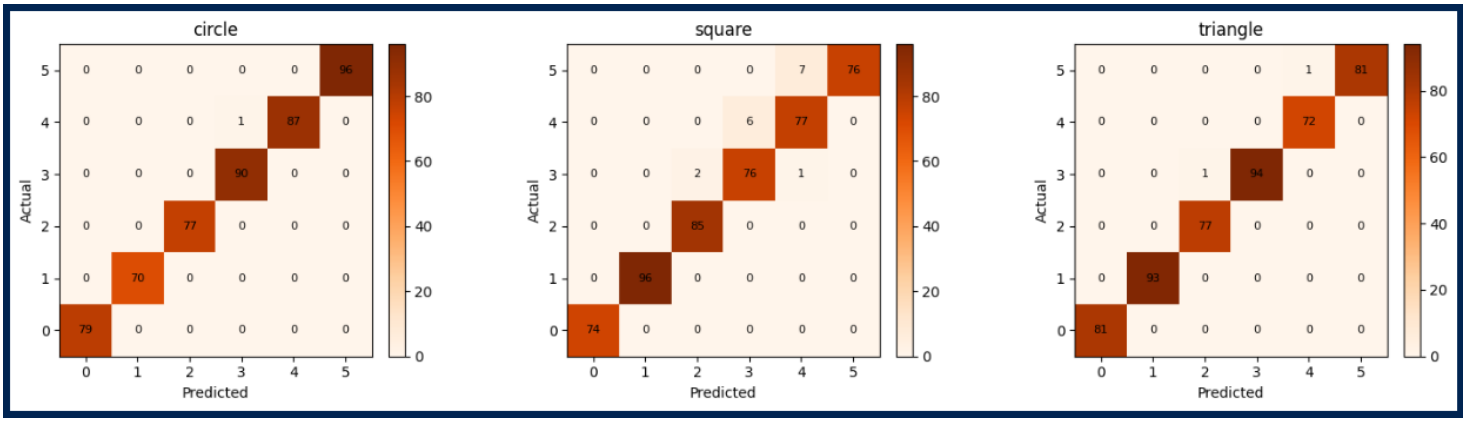
Learning Curves



Scatter Plot with value:frequency for each circle, square, triangle



Confusion Matrices for each shape of test set



Test set results

epochs: 100
5000 images 64x64

circle:	MAE=0.080	RMSE=0.112
square:	MAE=0.143	RMSE=0.206
triangle:	MAE=0.080	RMSE=0.120

Results from earlier runs

epochs: 100
3000 images 64x64:

circle: MAE = 0.115 RMSE = 0.156
square: MAE = 0.195 RMSE = 0.271
triangle: MAE = 0.105 RMSE = 0.165

epochs: 300
3000 images 64x64:

circle: MAE = 0.073 RMSE = 0.105
square: MAE = 0.126 RMSE = 0.193
triangle: MAE = 0.072 RMSE = 0.125

Summary of results

Results were mostly good. The network predicts circles and triangles very accurately (MAE ~ 0.08), and squares are okay but worse (MAE ~ 0.14–0.20). Most test images are predicted within ±1 count for each shape.

Successes

- Model learns quickly and gives low average error (small MAE/RMSE).
- Visual checks (scatter, confusion matrices, error histograms) show most predictions cluster near the true counts.
- Dataset generator and non-overlap logic make labels reliable, so training signal is clean.

Challenges

- Squares are harder to predict consistently with fewer epochs (25–50) performance on squares is noticeably worse.
- More epochs helped but too many can overfit; need a validation check to pick best epoch. 3000 images & 300 epochs return best results to date

Other issues

Limited shape size variety, synthetic simplicity may not transfer to real images, and overlapping/occlusion or extreme noise hurts accuracy.

Quick fixes to try

- Train longer with early stopping on validation loss (pick best epoch).
- Add more training images or augmentations (scale, rotate, noise).
- Slightly larger or deeper model, or initialize final bias to label means (already helpful).

Conclusion

Recap

Made a program that creates 64×64 grayscale pictures with circles, squares, and triangles and saves the true counts. Built and trained a small CNN (SmallCountNet) to predict three numbers: how many circles, squares, and triangles are in each picture. Split data into train/val/test, trained with Adam and L1 loss, saved the best model, and made plots (learning curve, scatter, confusion matrix, error histograms). Measured performance: low average error (MAE/RMSE), most predictions within ±1 count; squares are the weakest class.

Limits and what to do next (if we had more time)

- **Limits:** synthetic images are simple and may not match real photos; shapes have limited size/variation; squares are harder to learn; possible overfitting if you train too long.
- **Easy next steps**: add more images, add rotations/scale/brightness augmentations, vary sizes/colors, or let some overlap to increase realism.
- **Smarter fixes:** try a slightly bigger network, tune hyperparameters, use early stopping, or add regularization (dropout/weight decay).
- **Long-term:** test/transfer on real data (cells, stars, defects), or move from counting to full detection (localize each shape).

Bottom line

Good proof of concept - counts are accurate overall, but squares need more data/ epochs/tweaks for consistent improvement.

Note: addendum below shows full description of the SmallCountNet CNN utilized in the study.

Addendum

Description of SmallCountNet (for IMG_SIZE = 64)

1. **Input:** one small black-and-white picture (64×64 pixels).

2. **Block 1:** small filters scan the picture and pick up simple things like edges or bright spots. This produces 16 small "maps" (like 16 versions of the picture showing different features). A ReLU keeps only positive responses. A 2×2 max-pool shrinks each map in half (now 16 maps at 32×32).

3. **Block 2:** another filter layer finds more complex patterns from Block 1. Now there are 32 maps. ReLU, then pool shrinks to 32 maps at 16×16.

4. **Block 3:** one more layer makes 64 maps, then pool → 64 maps at 8×8.

5. **Summarize:** each of those 64 maps is averaged down to one number, so you end with a list of 64 numbers summarizing the whole image.

6. **Predict:** those 64 numbers go into a small fully connected network which outputs 3 numbers — predicted counts for circles, squares, and triangles. We usually round/clamp those to get integer counts.

7. **Training:** the model learns by minimizing the average absolute error between predicted and true counts (L1 loss). It's trained with the Adam optimizer (a fast, commonly used optimizer). The final layer's bias is set to the average counts at the start so training begins sensibly.