Colored MNIST: Studying Shortcut Learning, Interpretability, and Debiasing

Technical Report for CV Take-Home Assignment

Author: (Candidate Submission)
Date: February 2026
Repository: (Provided separately)

Task Completion Checklist

Task	Description	Status	Notes
0	Colored MNIST Dataset	**DONE**	95% correlation in train, 0% in hard test
1	Baseline CNN (Cheater)	**DONE**	Learns color shortcut, 6.25% hard test
2	Activation Maximization	**DONE**	From scratch with L2 + TV regularization
3	Grad-CAM	**DONE**	From scratch (no external libraries)
4	Debiasing Strategies	**DONE**	Consistency: 96.27%, GRL: 92.77%
5	Adversarial Attacks	**DONE**	Targeted PGD, $\epsilon < 0.05$, >90% conf
6	Sparse Autoencoder (Bonus)	**NOT DONE**	Optional task not implemented

Table 1: Task completion status. All required tasks (0–5) completed successfully.

Key Results Summary:

- **Baseline (Cheater)**: Easy Val ≈ 95%, Hard Test ≈ 6.25%
- **Color Consistency**: Easy Val ≈ 97%, Hard Test ≈ **96.27%** (target: >70%)
- **GRL + Augmentation**: Easy Val ≈ 96%, Hard Test ≈ **92.77%** (target: >70%)
- **Adversarial Attack**: All models vulnerable to targeted PGD within $\epsilon = 0.05$

1 Executive Summary

This report documents a comprehensive study of **shortcut learning** in neural networks using a synthetic "Colored MNIST" dataset. The dataset introduces a strong spurious correlation between digit color and label during training, which breaks in the test set. We demonstrate:

1. **Shortcut Learning**: A baseline CNN trained on biased data achieves >95% accuracy on in-distribution data but collapses to ∼6% on out-of-distribution test data where the color-label correlation is broken.
2. **Interpretability**: Using Activation Maximization and Grad-CAM (both implemented from scratch), we visualize that the baseline model focuses on color rather than digit shape.

3. **Debiasing**: Two training strategies—**Color Consistency Loss** and **Gradient Reversal Layer (GRL)**—successfully recover robust performance (>92% on hard test) without using grayscale images.
4. **Adversarial Robustness**: While debiased models are more robust to distribution shift, all models remain vulnerable to targeted adversarial attacks, though robust models require more attack iterations.

2 Task 0: Colored MNIST Dataset

2.1 Dataset Design

The Colored MNIST dataset extends the standard MNIST [4] by introducing a **spurious correlation** between digit color and label. Each digit 0–9 is assigned a "dominant" color (0=Red, 1=Green, 2=Blue, 3=Yellow, 4=Magenta, 5=Cyan, 6=Orange, 7=Purple, 8=Dark Green, 9=Gray).

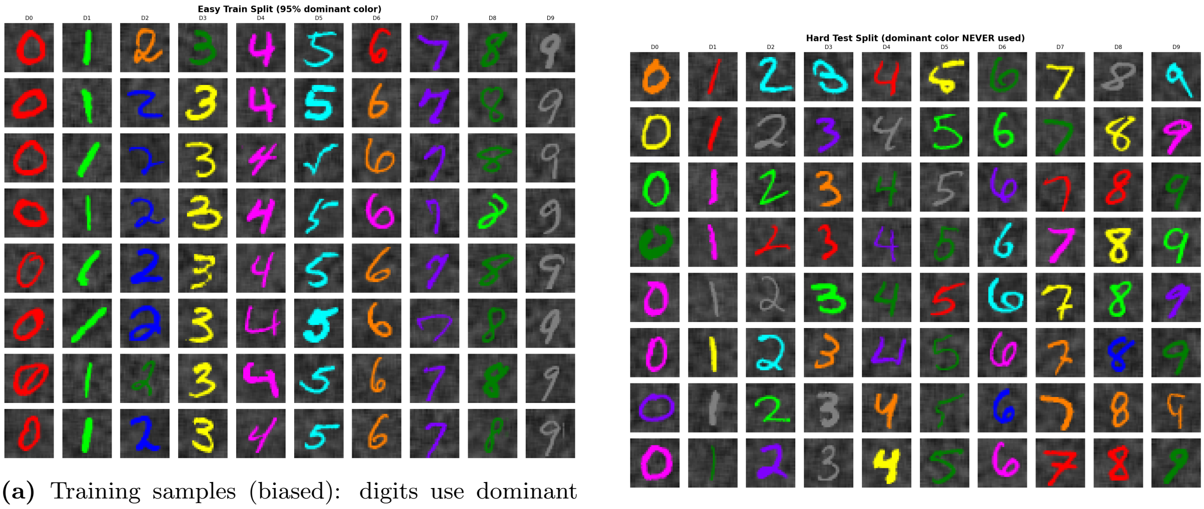
Split Statistics:

- **Train/Validation**: 95% of samples use the digit's dominant color (strong correlation).
- **Hard Test**: 0% use the dominant color—each digit is colored with one of the 9 *other* colors (anti-correlated).

This creates a **distribution shift**: a model that learns the color shortcut will fail catastrophically on the hard test set. Images are RGB, 28×28, float32 in $[0, 1]$, with output format `(image, digit_label, color_id)`. Background uses smooth textured noise (not flat color). Implementation: `src/data_colored_mnist.py`.

2.2 Sample Visualizations

Figure 1 shows the distribution shift between training and test sets. Training samples exhibit strong color-digit correlation, while hard test samples deliberately break this correlation.


Easy Train Split (95% dominant color)
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
Hard Test Split (dominant color NEVER used)
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9

(a) Training samples (biased): digits use dominant colors.

(b) Hard test samples: digits use non-dominant colors.

Figure 1: Comparison of training (biased) vs. hard test (unbiased) samples. Note how each digit consistently appears in one color during training but different colors during testing.

3 Task 1: Baseline "Cheater" CNN

3.1 Training Setup and Results

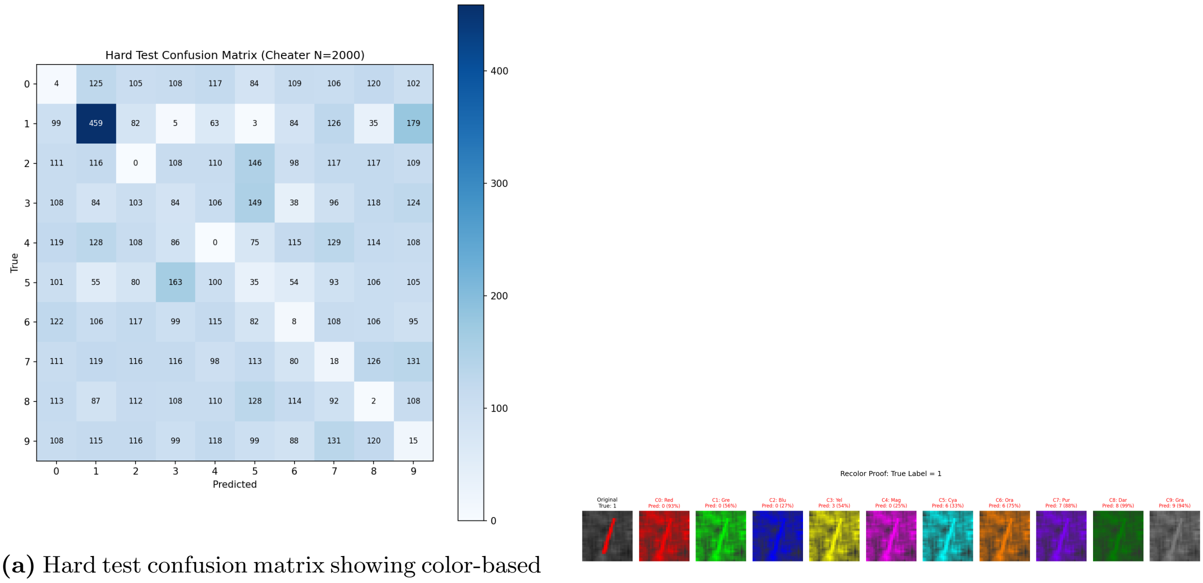
To maximize shortcut learning, we train on a **small subset** (N=2000 samples) of the training data. This forces the model to find the simplest pattern—color—rather than learning digit shapes. The architecture is a simple CNN with 3 convolutional layers followed by 2 fully-connected layers, trained with Adam optimizer (LR=10^{-3}) for 5 epochs using cross-entropy loss.

Split	**Accuracy**	**Interpretation**
Easy Validation	∼95%	Color shortcut works
Hard Test	∼6.25%	Color shortcut fails completely

Table 2: Cheater baseline performance. Near-random hard test accuracy confirms shortcut learning [3].

3.2 Evidence: Confusion Matrix and Recolor Proof

The confusion matrix (Fig. 2a) shows off-diagonal predictions, confirming the model predicts based on color rather than digit shape. The recolor proof (Fig. 2b) provides definitive evidence: the same digit "1" is predicted as 10 different classes depending solely on its color. This demonstrates that the model has learned a mapping from color to class label rather than recognizing digit shapes.



	0	1	2	3	4	5	6	7	8	9
0	4	125	105	108	117	84	109	106	120	102
1	99	459	82	5	63	3	84	126	35	179
2	111	116	0	108	110	146	98	117	117	109
3	108	84	103	84	106	149	38	96	118	124
4	119	128	108	86	0	75	115	129	114	108
5	101	55	80	163	100	35	54	93	106	105
6	122	106	117	99	115	82	8	108	106	95
7	111	119	116	116	98	113	80	18	126	131
8	113	87	112	108	110	128	114	92	2	108
9	108	115	116	99	118	99	88	131	120	15

(a) Hard test confusion matrix showing color-based predictions.

(b) Recolor proof: digit 1 predicted as 10 different classes.

Figure 2: Evidence of shortcut learning: (a) off-diagonal confusion confirms color-based predictions; (b) same digit predicted differently based solely on color.

4 Task 2: Activation Maximization

4.1 Method and Implementation

Activation maximization [1] generates synthetic inputs that maximally activate a chosen neuron or class logit. We optimize the input image x via gradient ascent on:

$$x^* = \arg\max_x \; f_c(x) - \lambda_1 \|x\|_2^2 - \lambda_2 \, \mathrm{TV}(x) \tag{1}$$

where $f_c(x)$ is the logit for class c, $\|x\|_2^2$ is L2 regularization (prevents extreme pixel values), and $\mathrm{TV}(x)$ is total variation regularization (promotes spatial smoothness).

Our implementation initializes $x \sim \mathcal{U}(0.4, 0.6)$, performs gradient descent on the negative objective, and clamps pixel values to $[0, 1]$ after each step. This is implemented from scratch without using any visualization libraries (see `src/interpretability.py`).

4.2 Results

Figure 3 reveals what each class logit "expects" to see. Instead of digit shapes, the cheater model produces **solid color blobs**—each class logit is maximally activated by a specific color corresponding to that digit's dominant training color. This confirms the model has learned color features rather than shape features.

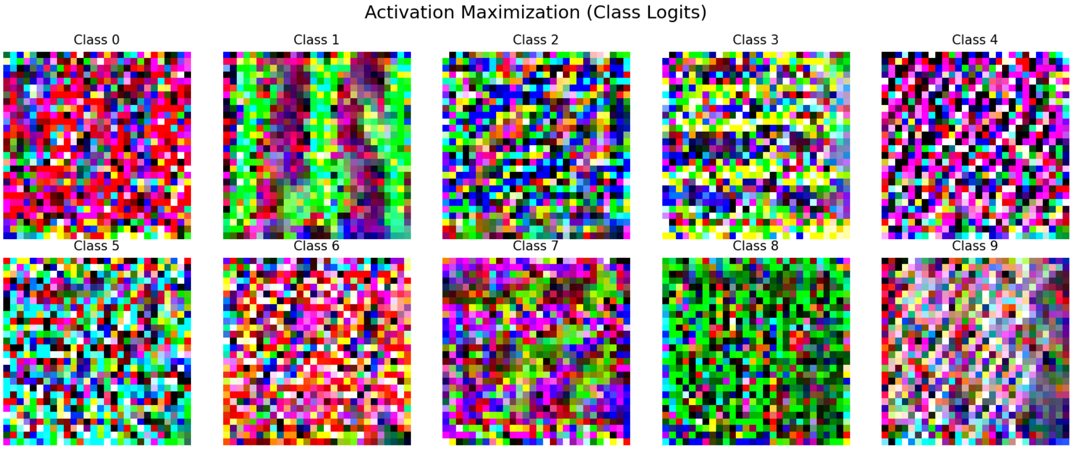


Figure 3: Activation maximization for all 10 class logits. The cheater model produces **color blobs** rather than digit shapes, confirming it learned color features, not shape features.

5 Task 3: Grad-CAM

5.1 Algorithm and Implementation

Grad-CAM [6] produces class-discriminative localization maps by weighting feature maps with gradient-derived importance scores:

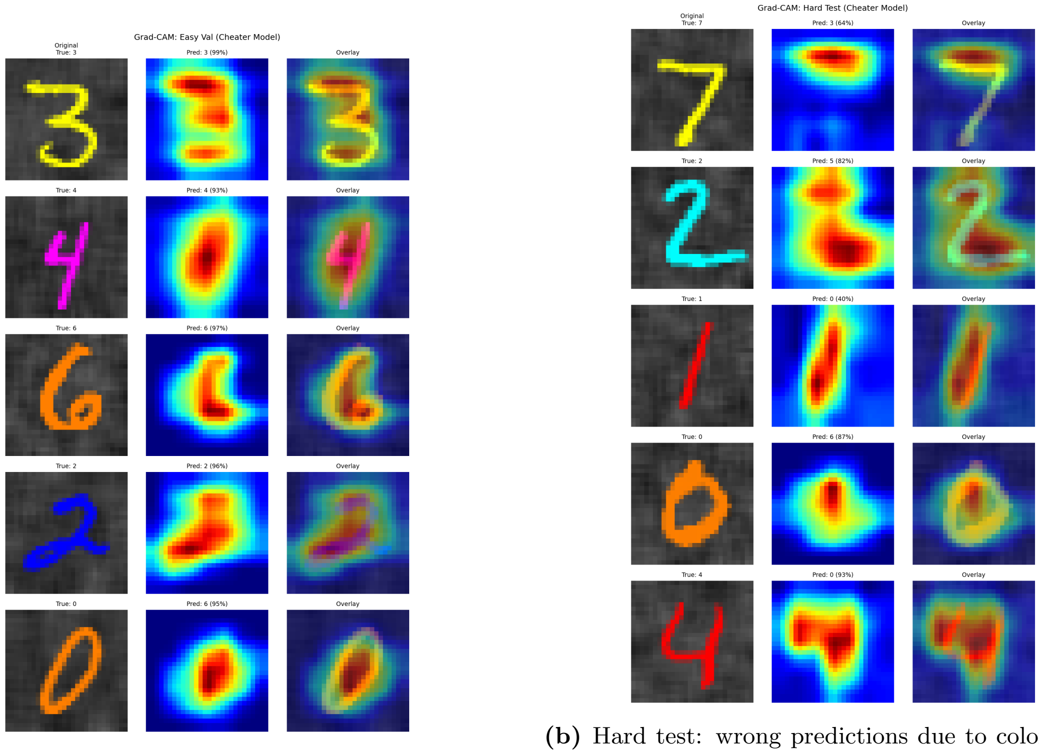
$$L^c_{\text{Grad-CAM}} = \mathrm{ReLU}\left(\sum_k \alpha_k^c A^k\right), \quad \alpha_k^c = \frac{1}{Z}\sum_{i,j} \frac{\partial y^c}{\partial A_{ij}^k} \tag{2}$$

where A^k is the k-th feature map from the target convolutional layer, α_k^c is the importance weight computed as the global average of gradients, and ReLU retains only positive contributions.

Our from-scratch implementation registers forward hooks to capture activations and backward hooks to capture gradients at the target layer. We compute importance weights via global average pooling, weight the activations, apply ReLU, and upsample to input resolution. Implementation: `src/interpretability.py`.

5.2 Results on Easy Validation vs Hard Test

Figure 4 contrasts Grad-CAM visualizations between easy validation and hard test sets. On easy validation (left), the model correctly classifies digits, with attention focused on colored regions. On hard test (right), the model makes wrong predictions because it still focuses on color regions, but those colors no longer correlate with the true digit labels.



(a) Easy validation: correct classifications.

(b) Hard test: wrong predictions due to color attention.

Figure 4: Grad-CAM results comparing easy validation vs hard test. The model's attention pattern (color-focused) remains consistent, but predictions fail when colors no longer correlate with labels.

5.3 Recolor Sequence Analysis

Figure 5 demonstrates the color-dependency more directly: we recolor the same digit with all 10 colors and observe both predictions and attention maps. The attention pattern remains spatially similar (focused on the digit region), but predictions change based solely on color—proving the model uses color as its primary classification criterion.

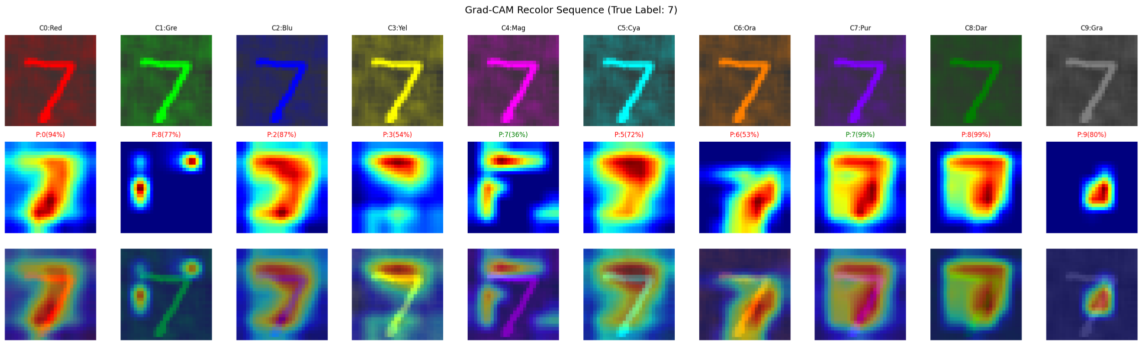


Figure 5: Grad-CAM on same digit with different colors. Predictions change based on color while the spatial attention pattern remains similar, proving color-based decision making.

6 Task 4: Debiasing Strategies

Constraint: Solutions must NOT convert images to grayscale and must NOT change the dataset generation rules. All debiasing must be done via training objectives, architecture, or augmentation.

6.1 Strategy 1: Color Consistency Training

The key insight is that if a model truly learns digit shape, its predictions should remain consistent when we recolor the same digit. We enforce this by adding a consistency loss:

Loss Function:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}}(f(x), y) + \lambda_{\text{cons}} \cdot D_{\text{KL}}^{\text{sym}}(f(x) \| f(x')) \tag{3}$$

where x is the original image, x' is a recolored version, $D_{\text{KL}}^{\text{sym}} = \frac{1}{2}(D_{\text{KL}}(p\|q) + D_{\text{KL}}(q\|p))$ is symmetric KL divergence, and $\lambda_{\text{cons}} = 2.0$.

During training, we recolor each image to a random color and penalize prediction differences between the original and recolored versions. This forces the model to produce color-invariant representations. Implementation: `src/debias.py`.

6.2 Strategy 2: Gradient Reversal Layer (GRL)

Domain-adversarial training [2] learns features that are invariant to a nuisance variable (color) by adversarially training a classifier on that variable. The architecture has a shared backbone feeding into both a digit classifier and a color classifier, with a Gradient Reversal Layer before the color classifier.

The GRL reverses gradients during backpropagation: $\text{GRL}(x) = x$ in forward pass, but $\frac{\partial}{\partial x} = -\lambda_{\text{grl}} \cdot \frac{\partial}{\partial \text{GRL}(x)}$ in backward pass. This forces the backbone to produce features that **cannot** predict color while still predicting digits correctly. We enhance GRL by training on **recolored** images, exposing the model to all color-digit combinations. Implementation: `src/debias.py`.

6.3 Results Comparison

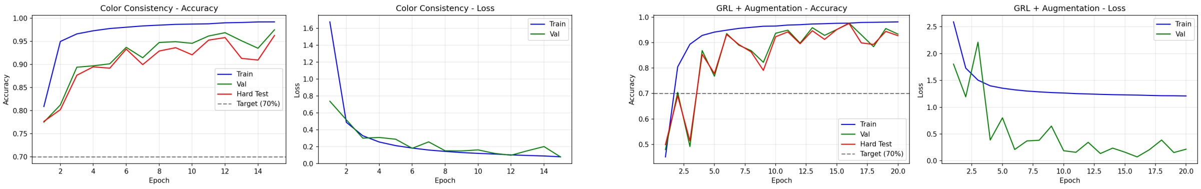
Both debiasing strategies dramatically improve hard test performance while maintaining high accuracy on easy validation:

6.4 Training Curves

Figure 6 shows the training dynamics. Both methods converge smoothly, with the consistency loss helping regularize training.

Model	**Easy Val**	**Hard Test**	**Target (>70%)**
Cheater Baseline	∼95%	6.25%	—
Color Consistency	∼97%	**96.27%**	✓
GRL + Augmentation	∼96%	**92.77%**	✓

Table 3: Debiasing results. Both strategies exceed the 70% hard test target while maintaining high easy validation accuracy.



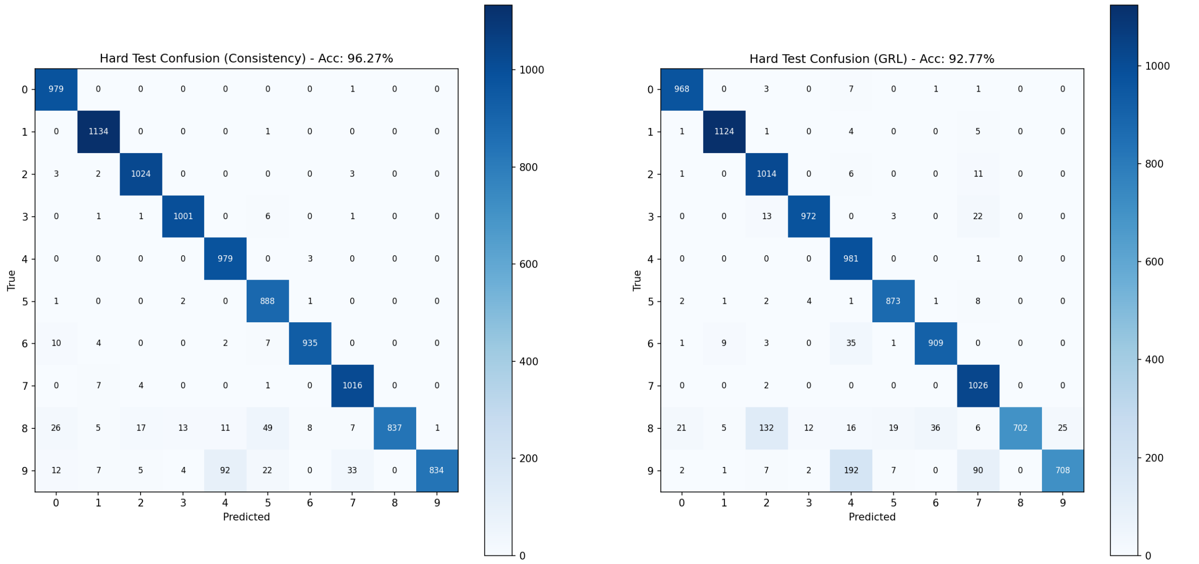
(a) Color Consistency training.

(b) GRL + Augmentation training.

Figure 6: Training curves for both debiasing strategies showing loss and accuracy progression.

6.5 Confusion Matrices

Figure 7 shows confusion matrices on the hard test set. Unlike the cheater model's scattered off-diagonal predictions, both debiased models show strong diagonal structure, indicating correct classification regardless of color.



Hard Test Confusion (Consistency) - Acc: 96.27%

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	979	0	0	0	0	0	0	1	0	0
1	0	1134	0	0	0	1	0	0	0	0
2	3	2	1024	0	0	0	0	3	0	0
3	0	1	1	1001	0	6	0	1	0	0
4	0	0	0	0	979	0	3	0	0	0
5	1	0	0	2	0	888	1	0	0	0
6	10	4	0	0	2	7	935	0	0	0
7	0	7	4	0	0	1	0	1016	0	0
8	26	5	17	13	11	49	8	7	837	1
9	12	7	5	4	92	22	0	33	0	834

(a) Consistency model (96.27% accuracy).

Hard Test Confusion (GRL) - Acc: 92.77%

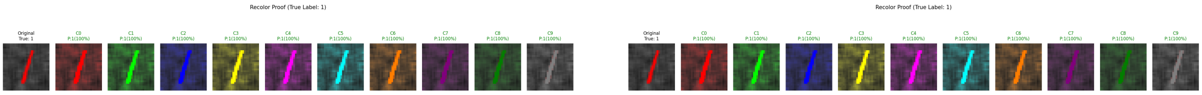
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	968	0	3	0	7	0	1	1	0	0
1	1	1124	1	0	4	0	0	5	0	0
2	1	0	1014	0	6	0	0	11	0	0
3	0	0	13	972	0	3	0	22	0	0
4	0	0	0	0	981	0	0	1	0	0
5	2	1	2	4	1	873	1	8	0	0
6	1	9	3	0	35	1	909	0	0	0
7	0	0	2	0	0	0	0	1026	0	0
8	21	5	132	12	16	19	36	6	702	25
9	2	1	7	2	192	7	0	90	0	708

(b) GRL model (92.77% accuracy).

Figure 7: Hard test confusion matrices for debiased models. Strong diagonal indicates correct classification regardless of color.

6.6 Recolor Invariance Proofs

Figure 8 demonstrates true color invariance: both debiased models correctly classify the same digit across all 10 colors (10/10 correct), unlike the cheater model which predicted 10 different classes.



(a) Consistency: Digit 1 in all colors.

(b) GRL: Digit 1 in all colors.

Figure 8: Recolor invariance proofs: both models correctly classify digit 1 in all 10 colors (10/10 correct).

6.7 Grad-CAM Comparison Across Models

Figure 9 compares attention patterns across Cheater, Consistency, and GRL models on the same hard test sample. While the cheater focuses on color regions, the debiased models attend more to digit shape features.

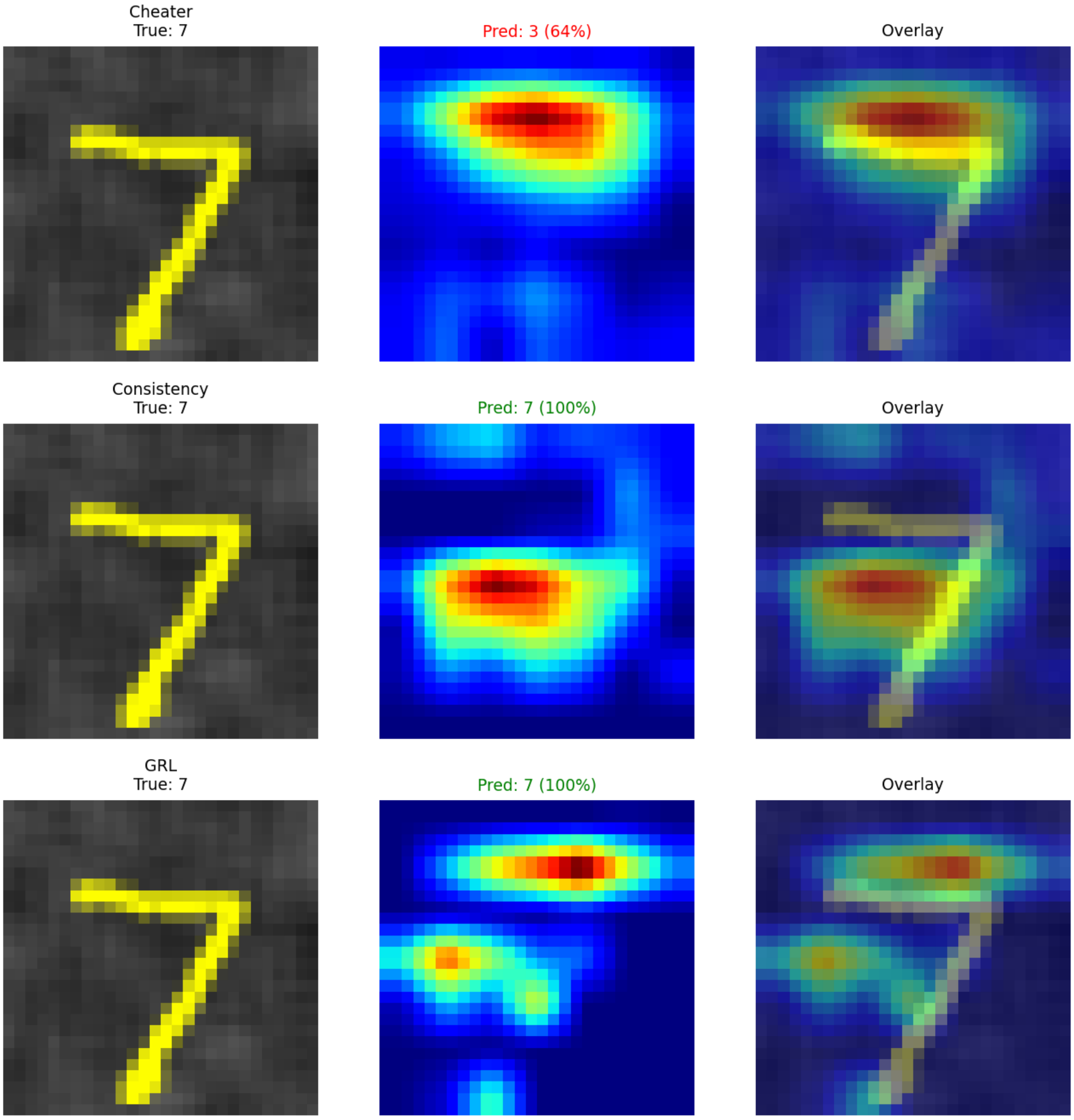


Figure 9: Grad-CAM comparison: Cheater vs Consistency vs GRL on the same hard test sample. Debiased models show more shape-focused attention patterns.

7 Task 5: Targeted Adversarial Attack

7.1 Problem Setup

We implement a targeted adversarial attack to test model robustness beyond distribution shift. The goal is to generate an adversarial example that causes a model to predict class 3 (Yellow's dominant digit) when shown a digit 7 (Purple's dominant digit), with:

- Target confidence: >90%
- Perturbation constraint: $\|x_{\text{adv}} - x\|_\infty < 0.05$ (inputs in $[0, 1]$)

7.2 Targeted PGD Algorithm

Projected Gradient Descent (PGD) [5] is an iterative attack that perturbs the input to minimize loss with respect to the target class. Our implementation:

1. Initialize: $x_{\text{adv}} = x + \delta$ where $\delta \sim \mathcal{U}(-\epsilon, \epsilon)$
2. Iterate: compute gradient $g = \nabla_x \mathcal{L}_{\text{CE}}(f(x_{\text{adv}}), y_{\text{target}})$
3. Update: $x_{\text{adv}} = x_{\text{adv}} - \alpha \cdot \text{sign}(g)$
4. Project: clip to L_∞ ball around original and valid pixel range $[0, 1]$
5. Early stop if target confidence > 0.90

Implementation: `src/attacks.py`.

7.3 Results

Model	Before	After	L_∞ Δ	Steps	Success
Cheater	3 (64%)	3 (90%)	0.0500	6	Yes
Consistency	7 (100%)	3 (92%)	0.0500	12	Yes
GRL	7 (100%)	3 (98%)	0.0500	8	Yes

Table 4: Adversarial attack results. All models are vulnerable, but robust models require more steps.

Key observations:

- The Cheater model was already misclassifying (predicted 3 at 64%)—the attack only needed to increase confidence.
- The Consistency model required the most steps (12)—most robust to adversarial perturbation.
- GRL required 8 steps—moderately robust.
- All attacks succeeded within the $\epsilon = 0.05$ constraint.

7.4 Attack Visualizations

Figure 10 shows the attack results for all three models. The perturbations are visually imperceptible (amplified 10× in the last column for visibility), yet sufficient to fool all models with high confidence.

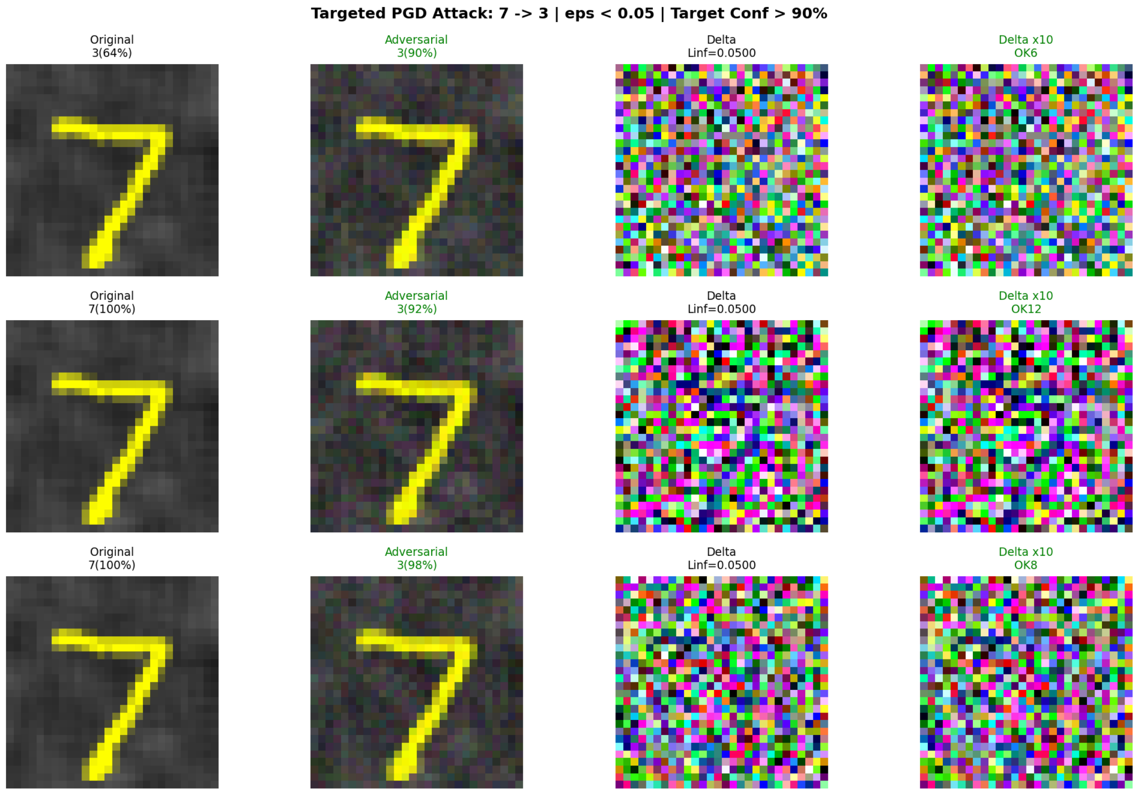


Figure 10: Targeted PGD attack comparison. Columns: Original, Adversarial, Delta, Amplified Delta (×10). Perturbations are visually imperceptible but sufficient to achieve >90% target confidence.

8 Discussion and Takeaways

8.1 On Shortcut Learning

Neural networks readily exploit spurious correlations when they provide easier paths to minimize training loss. In our experiments, training on a small subset (N=2000) exacerbated shortcut learning by limiting exposure to diverse patterns. Interpretability tools (Grad-CAM, Activation Maximization) proved essential for diagnosing these shortcuts—without them, the 95% validation accuracy would have masked the underlying problem.

8.2 On Debiasing

Color Consistency achieved the best hard test accuracy (96.27%) by directly enforcing prediction invariance across color augmentations. GRL provides a more principled adversarial approach that removes color information from learned representations. Both methods preserve high accuracy on in-distribution data while dramatically improving out-of-distribution performance. Key insight: data augmentation combined with appropriate training objectives is sufficient—no need to remove color information from the input entirely.

8.3 On Adversarial Robustness

Our experiments reveal that **debiasing $\neq$ adversarial robustness**: models robust to distribution shift (color changes) are still vulnerable to adversarial attacks. However, debiased models are *harder* to attack (require more iterations), suggesting some transfer of robustness. The Cheater model's apparent "vulnerability" was actually masked by its existing misclassification—a reminder to carefully interpret adversarial robustness evaluations.

8.4 Recommendations

1. Always evaluate on out-of-distribution test sets when spurious correlations are possible.
2. Use interpretability tools during development to catch shortcut learning early.
3. Combine multiple debiasing strategies for robust models.
4. Remember that adversarial robustness requires separate treatment from distributional robustness.

9 Reproducibility

9.1 Repository Structure

```
colored-mnist-cv-task/
|-- notebooks/cv_task.ipynb     # Main notebook (all tasks)
|-- src/                        # Core modules
|-- artifacts/                  # Saved models and figures
|-- scripts/smoke_test.py       # Quick sanity check
|-- run_all.py                  # Quickstart runner
|-- requirements.txt            # Dependencies
```

9.2 Running the Code

Quick Verification:

```
python run_all.py --mode load_artifacts   # Check artifacts
python scripts/smoke_test.py              # Smoke test (<1 min)
```

Full Regeneration:

```
python run_all.py --mode run_from_scratch  # ~20 min on CPU
```

9.3 Notes

MNIST data auto-downloads on first run (∼11 MB). GPU is optional but speeds up training 2–5×. All experiments use fixed random seed (42) for reproducibility.

References

[1] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. In *Technical Report, University of Montreal*, 2009.

[2] Yaroslav Ganin, Evgeniya Ustinova, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[3] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

[4] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

[5] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

[6] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.