PreCog

Task 0

https://www.youtube.com/watch?v=1cq-jXyi6ek
imagnet texture bias and shape bias, cnn naturally texture biased. more robust to noise even without showing noisy data during training.

so first we need to create a color map to map the numbers. the colors are represented in float (1.0 being the equivalent for 256) since neural networks function better with smaller numbers.

multiple ways of coloring:
bg/number/texture.

bg is too simple (model looks only at top-left pixel) and not allowed. we need the model to look both at the digit and the color. texture, made by multiplying random noise could also be used (similar to how a cnn visualises grass or fur) but will make it much harder for the model to learn and to visualise using a heatmap in gradcam as well. smth like ResNet-18 (which we use later) might actually struggle to memorize random noise patterns and decide that its easier to just learn the shape (texture bias is introduced, which could counteract the color bias). therefore the number itself (foreground) is colored instead.

coloring the dataset itself:
1. download the MNIST dataset if absent.
2. load the img and label.
3. convert the image to a tensor.
4. 95% of the time, assign it the right color code when in train mode (signified by self.train). else if in test mode or in train with 5% error, assign a random color (ENSURE that the color code doesn't match at any cost by reassigning it till the case is met).

> → we are using a dynamic probability here instead of a direct 95/5 split of the data. this is to ensure variation. else the model might just memorise that a specific version of a number is colored with a specific color

5. match the color code to the color_map in order to get the color in float notation.
6. color the image by broadcasting. reshape the color tensor to (3,1,1) to align with the image (1,28,28) by using .view(3,1,1) and then broadcast it by

multiplying.
7. return the result

Task 1
https://www.geeksforgeeks.org/deep-learning/resnet18-from-scratch-using-pytorch/
https://www.kaggle.com/code/ivankunyankin/resnet18-from-scratch-using-pytorch/notebook

now the task is to train a lazy convolutional neural network on our train data set. this can be done in two ways:
1. using ResNet-18
2. using simple 3-Layer CNN

both have their own reasons to be used:
→ simple 3- layer CNN has very few parameters, thereby making it computationally poor. this forces it to learn something simple like color over say, edges/ shapes. this showcases spurious correlation more clearly. we know that if it fails, its clearly due to the bias we instilled. but in the case of ResNet-18, it could also technically be due to the noise from other layers. it also preserves spatial information clearly, allowing us to see make out distinctions from the gradcam heatmaps easily.
→ ResNet-18 is a high capacity, state-of-the art architecture. by observing that if even it faces issues with this bias, it tells us that this problem is not unique to simple models but deep learning in general.

initially, i've decided to use ResNet-18 for reasons stated above, even if the end gradcam heatmaps were to look messy either way as spatial information is compressed. the reason for this is as follows:

1. we can prove that even smart models with many parameters fall prey to this bias. and not just because a model is simple.
2. allows us to observe how 'better' models can sometimes be 'worse' at generalization when data is flawed
3. color is technically a "zero-frequency texture"; deep architectures naturally prefer local information (like color) over global information (like the shape of a number)
4. simple 3-layer cnns aren't used in practice

for this specific task, we can't use the standard ResNet-18 directly for the following reasons:

1. it is designed for ImageNet (224×224 images), which get compressed aggressively in the first stage so that they don't end up taking too much memory.
2. through standard ResNet layers:

Layer / Stage	Standard ImageNet Input (224×224)	MNIST Input (28×28)	Issues
Input Image	224×224	28×28	input data is 8x smaller
Conv1 (Stride 2)	112×112	14×14	stride 2 conv essentially makes it halved
MaxPool (Stride 2)	56×56	7×7	after the maxpooling of stride 2 again, we get an image that's only 7×7 (blurry)
Layer 1	56×56	7×7	size remains const.
Layer 2 (Stride 2)	28×28	4×4	MNIST loses almost all detail
Layer 3 (Stride 2)	14×14	2×2	ImageNet is a thumbnail. MNIST is 4 pixels.
Layer 4 (Stride 2)	7×7	1×1	MNIST data is now just a pixel
AvgPool	1×1	1×1	global average pooling has no effect on MNIST data

3. therefore we need to ensure that the MNIST data doesn't become too small towards the end.

now, in order to make it work, the following changes must be made:

1. since our starting images are already small, we can make the first step much more gentler → 3×3 kernel with stride 1 (instead of 7×7, stride 2)

2. delete the MaxPool layer so we get the full 28×28 size image for our forward pass
3. modify the output layer so that we have 10 outputs for our classifier
4. also ensure that weights are 0 so we start fresh instead of using imagenet ones.

however upon practical testing, we recognised that the model even with the bias was very robust:

```
Starting Training...
Epoch 1: Train 99.4% | Test 96.4%
Epoch 2: Train 99.6% | Test 96.8%
Epoch 3: Train 99.6% | Test 96.2%
Epoch 4: Train 99.7% | Test 97.1%
Epoch 5: Train 99.8% | Test 98.1%
```

reducing the kernel size would make it essentially blind even though it would help it learn colors easily.

→ decided to completely tweak my design approach. tried creating a simple 3 layer cnn model with tiny filters.

Task 4

That is the exact right question to ask. In these types of "AI Psychology" tasks, the constraints are the puzzle. If you just fix the data, you aren't fixing the model's behavior—you're just removing the trap.

Here is the breakdown of the "Rules of Engagement" for Task 4, and then a verdict on your 8 ideas.

The Hard Constraints (What you CANNOT touch)

1. **The Raw Dataset Source:** You cannot download ImageNet, CIFAR, or even the original unbiased MNIST. You **must** use the `ColoredMNIST` class you wrote, where 95% of 0s are Red.
2. **Grayscale Conversion:** The prompt explicitly bans "converting the image to grayscale." This is because grayscale is a "cheat code" that physically removes the bias dimension. The goal is to make the model *choose* to ignore color, not to blind it.
3. **The Labeling:** You can't change the ground truth labels (obviously).

The Playground (What you CAN change)

Everything else is fair game. You are hacking the **learning process**, not the data.

- **The Loss Function:** (e.g., penalize the model if it focuses on color).
- **The Sampler:** (e.g., change *how often* the model sees the 5% "rebel" images).
- **The Architecture:** (e.g., split the network into two heads).
- **The Optimizer:** (e.g., distinct learning rates).
- **The Augmentation Pipeline:** (With caveats, see below)

PyTorch