Project Report (The Lazy Artist)

Vikesh Kansal

0. Tasks Attempted (& Why)

- I have attempted tasks 0 to 5 of the CV project.
- I have not attempted task 6 due to a lack of time.

1. Introduction

- **Goal**: The goal of this project was to train a Convolutional Neural Network (CNN) on a biased dataset (Colored MNIST), demonstrate its failure on conflicting data, analyze the internal mechanism of this failure using interpretability tools, and implement methods to mitigate it.
- **Dataset (Colored MNIST)**:
  - **Construction**: In the training set, digit color is highly correlated with the digit class (for example 0 is always red), however this is only for 95% of the digits; for 5% of the digits, this correlation is randomized (for example, a 0 may be a green). In the "conflicting" test set, this correlation is randomized for all of the data.
  - **Problem**: Models tend to take the shortcut path. They learn the much more prevalent (and easy) feature of color over shape. We need to see why this is happening, what the model is doing under-the-hood, and how we can fix it.

2. Task 0 & 1: The Baseline Model and Bias Quantification

- **Dataset Generation**:
  - I made three datasets; `train_colored` and `test_colored`, `test_cycled`, and the normal, uncolored dataset but with three channels (`greyscale`) for future use.
  - Here, `train_colored` is the biased dataset, `test_colored` is the dataset with randomized backgrounds.
  - `cycled` was a dataset I made to test that the model is actually looking at color by just cycling the color that is assigned to each digit; i.e. 0 is assigned 1s color, 1 is assigned 2's color, and so on.
  - The backgrounds aren't just flat color; for each image, random noise is generated first, which is then multiplied by the color assigned to the digit.
- **Model Architecture**:
  - `BaselineNet` is a simple CNN consisting of 3 Convolutional blocks (each with Conv2D, GroupNorm, and ReLU), followed by MaxPooling, and a Global Average Pooling layer feeding into a linear classifier (10 outputs).

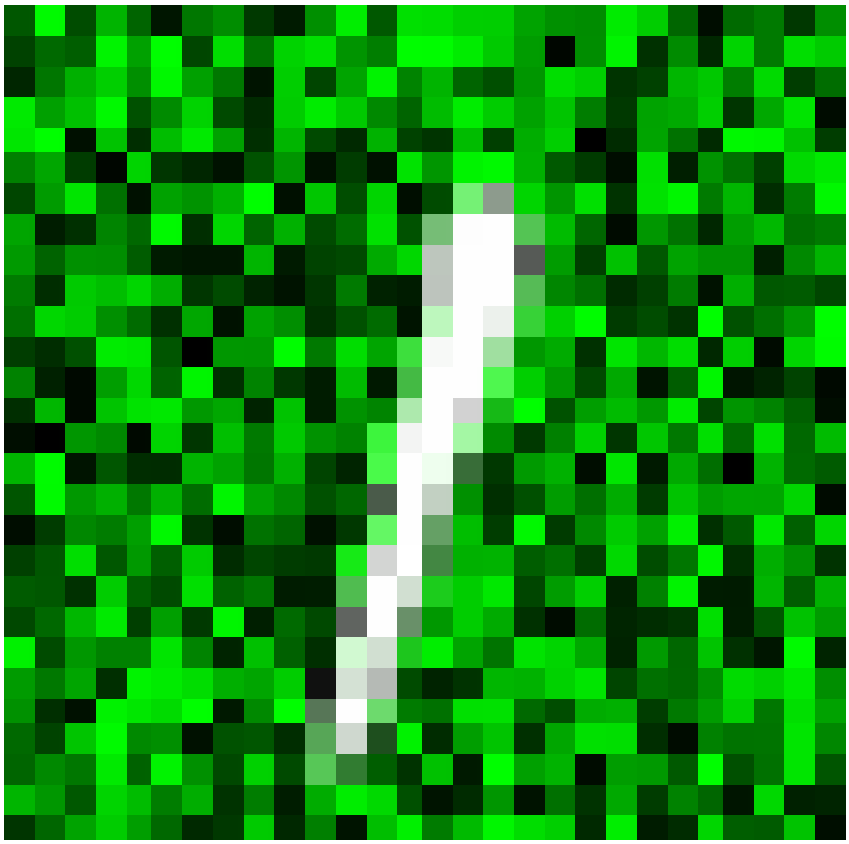

Figure 1: Example Image

  - This model architecture was picked with a lot of trial and error; specifically with trying to get the model's accuracy to be below 20%, and considering we want the model to take the shortcut path.
  - One thing I noticed with this was the sensitivity towards the kernel size and how much things like normalization matters; initially I was using residual blocks, and had ∼100k parameters. I was getting ∼90% accuracy even on the test dataset. I then reduced the parameters, removed BatchNorm, and replaced the 3x3 sized kernel; replacing them with simpler components and trying different combinations was a big part of the trial-and-error process.

- **Hyperparameters**:
  - Optimizer: SGD
  - Learning Rate: 0.1
  - Batch Size: 16
  - Epochs: 10
  - These hyperparameters were also picked mainly with trial and error.
- **Performance**:
  - **Training Accuracy**: **∼95%**
  - **Conflicting (Hard) Test Accuracy**: **3.77%**
  - **Interpretation**: The model achieves near-perfect training accuracy but fails com-

pletely on the hard test set (worse than random guessing's 10%), indicating it has learned to rely solely on the spurious color correlation.

- **Confusion Matrix**:

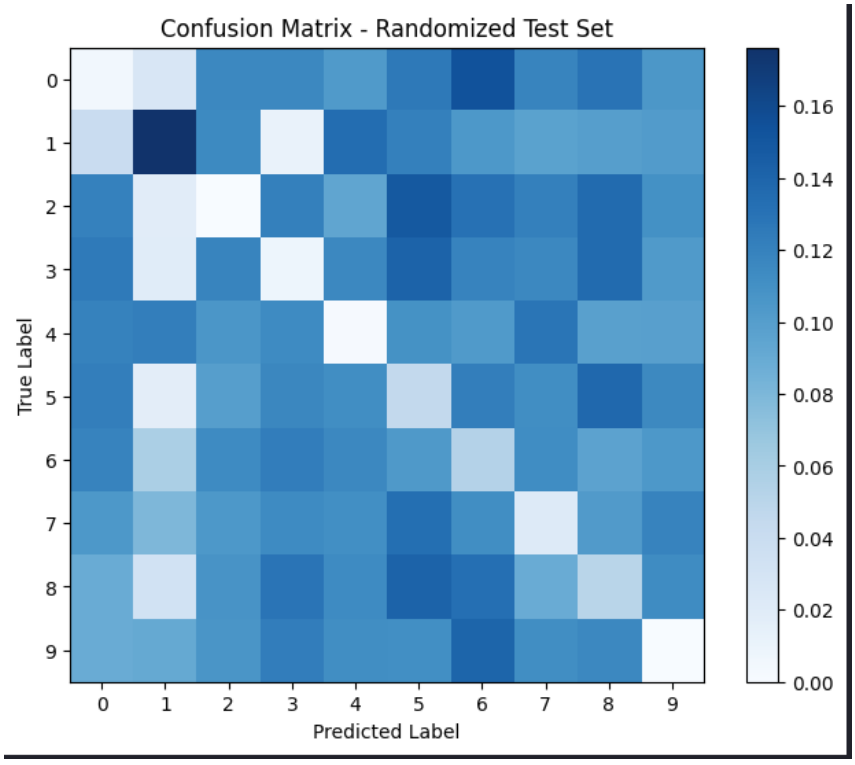

Figure 2: Confusion Matrix

  - **Analysis**: The confusion matrix (made for the test set) reveals that the model is mainly looking at color since it's very distributed.
  - One thing I noticed was that the digit 1 specifically normally had low confusion (i.e. it was correctly classified) a lot more than the others, but still low compared to an absolute scale. I suspect that this is because the digit 1 is just a vertical line, and so it is a lot easier for the model to identify it, compared to 3, for example, which is made up of curves.
  - Something else to note is that the confusion matrix has very low confidence values on the diagonal (i.e. at any index (i, i)), since the test set has backgrounds of anything but the color assigned to a digit.

3. Task 2: Interpretability - "What is the model looking at?"

Feature Visualization (Activation Maximization): I tried several things for this:

1. **Memory & Optimization Experiments**:
   - The first step consisted of trying to see what a model remembers; I passed in images belonging to each class, captured the activations of the neurons in the 3rd convolutional layer (I'll refer to this as c3) and trained a tensor such that the activations of the tensor's c3 match the captured c3.
   - The next thing I tried was to just optimize image tensors to produce each class' ideal

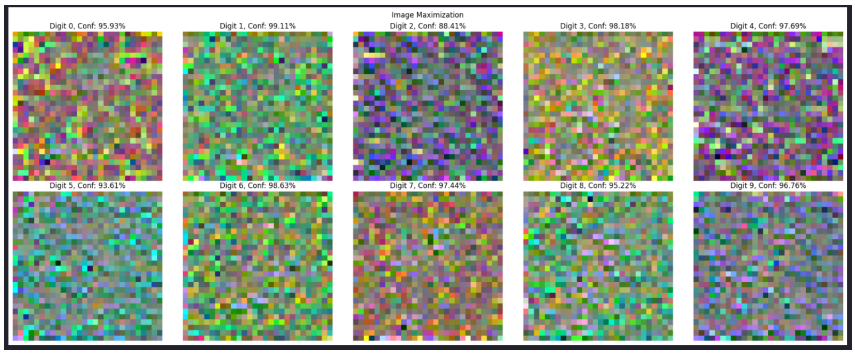

Figure 3: Seeing what a model remembers - maximizing values for minimal c3 activation difference

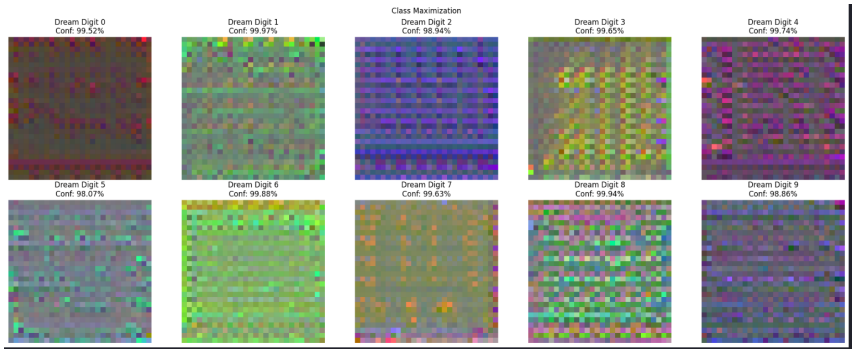

Figure 4: maximizing values for each class' probability

image, by maximizing their probabilities. To this, I tried starting off with random noise, and also tried starting off with the background texture that I made the dataset with.

- As an example, if I was generating the image for 1 then I'd choose the background texture for 1, and optimize from that. The main thing I noticed with this is that the model simply changed around the pixels to simply maximize what I had asked for; not necessarily in a way that would be interpretable to humans.
- From here I realized the importance of regularization and tried applying it, however I didn't get concrete outputs with that and left it at that.

2. **Model Simplicity Analysis (SLT, PCA, Ablation & NMI)**:
   - The next thing I decided to do was to show that the model is actually really simple and that its dimensionality isn't very high. The main way I decided to do this was using 3 things - Singular Learning Theory (specifically a package developed by Timaeus - `devinterp` helped me with this, and I'll mention SLT later on in the report), Principal Component Analysis (PCA), Neuron Ablation (which measures how much the accuracy drops when a neuron is essentially not considered), and a (Normalized) Mutual Information graph.
   - The neuron ablation was done on the output of the GAP layer in my model (there are 16 neurons). This was chosen since it's the layer right before the final classification layer, and so I believe that it is the most likely to be a measure of human-measurable features; and hence the most important one to look into. Similarly the NMI graph was also made using these same 16 neurons.

- The output I got for this was like so:
  - **LLC**: The Local Learning Coefficient came to be around 1.5.

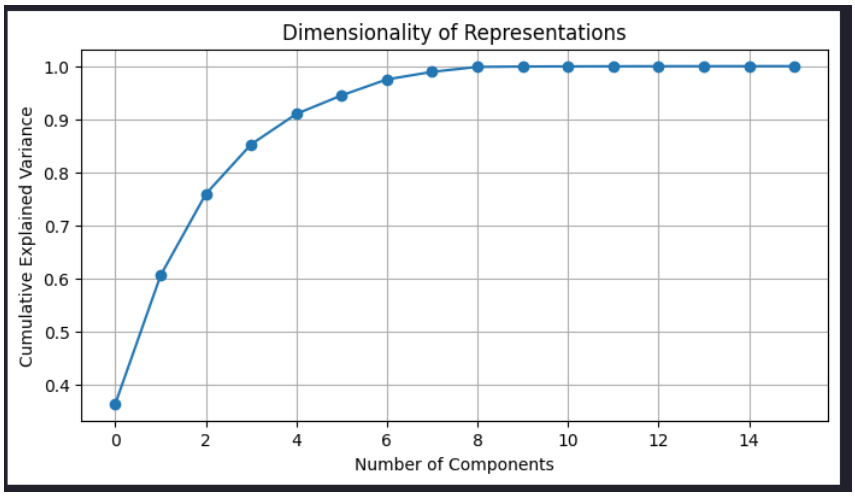

Figure 5: PCA

  - **PCA**: Through PCA, the number of components came out to be around ∼9-10.
  - **Ablation**: Through the ablation I identified 9 neurons which cause the accuracy to drop by more than 1% (i.e. they're significant). Through PCA and this, especially showed that the model was simple; only these neurons really mattered.

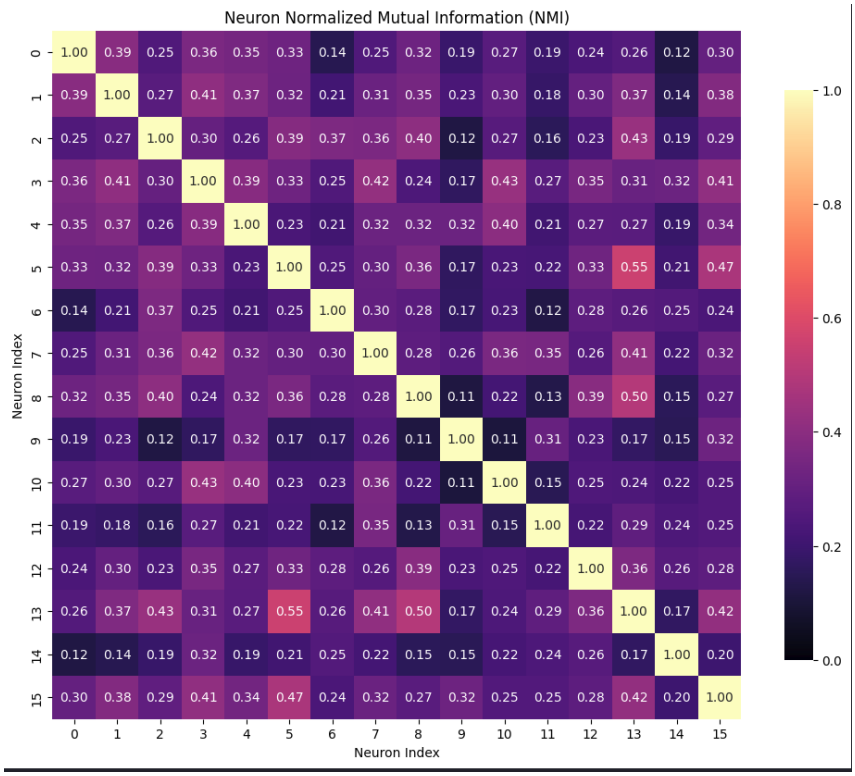

Figure 6: Normalized Mutual Information

  - **NMI Graph**: Through the NMI graph I found out about the complete relations

(both linear and non-linear which is why I picked NMI) between the neurons. Also I learnt about this recently and was looking where to apply this to. This combined with later steps gives a lot of insight into the model.

- All of this was implying that the model was clearly very simple in nature. The ablation is also used a lot in future steps.

3. **Neuron Grouping & Analysis**:

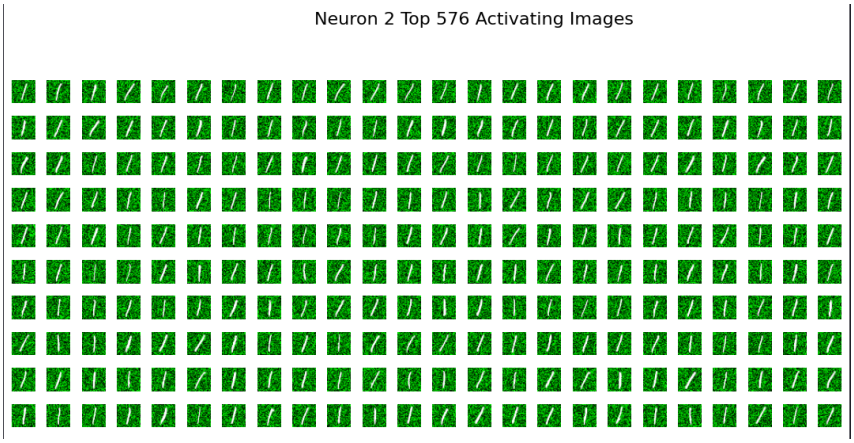

Figure 7: Neuron 2's Top-K images

- Next, I tried grouping the neurons based on how they get activated. For this, I first found the top 'k' (here k is 576) images that activate each of the 16 neurons and stored references to each of these images.
- Once I found this, I saw that I could infer a lot about the neurons by this itself; for example, Neuron 2 had all Green 1s. Neuron 8 had all Green 8s, and so on. I came up with 2 methods to break down what a specific neuron is doing. I have chosen Neuron 2 for the following methods since Neuron 2's ablation score was one of the highest; also the top-k images were just green 1s which should give somewhat of an idea of what it represents, so that we can validate the methods (by checking if what we get aligns with the clearly common features) and also seeing if we can interpret more about the seemingly straightforward neuron.

Method 1: Vision-Language Models (CLIP/BLIP)

- Using the same top-k images, we extract embeddings and semantic descriptions using a vision-language model (e.g., CLIP). We perform similarity analysis and clustering in semantic space to identify human-aligned modes of activation.
- Then, we look at the images using BLIP, and see how a human would describe the images. We collect the most occuring keywords and form sentences from the keywords to see how well they stack up against the centroid of all the vectors whose embeddings we generated from the top k images.
- *Result*: That didn't work; I think that the vision encoder produces embeddings that are highly specific; also since the mean of the embeddings is very high, that means that they are pointing in a very specific direction. Since the vector space is 512-dimensional, it can be very hard to get a good match. Although, some text embeddings had a noticable match with the embeddings as compared to the others, the similarity was still not very high.

Method 2: Visual Snippets & Activation Injection

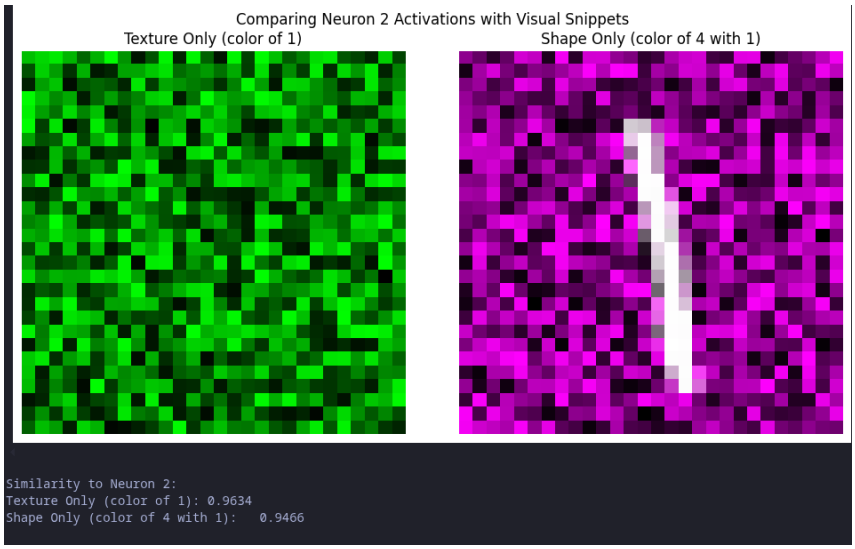

Figure 8: Example of the visual snippet (with vector-embedding similarity scores)

- The next thing I tried doing was instead of comparing it with human-interpretable text, compare it with 'visual snippets'. For example, I could compare it with just an entire 28x28x3 image of the background texture, or an image which has the 'anti-color' as the background with the number 1.
- This did reveal more about the neurons (for example, for Neuron 2, I found that it was activated most by green textures, and blue textures decrease the activation; this was consistent with the fact that Neuron 2 had all Green 1s in its top-k images). However just doing vector similarity comparisons didn't give a lot of information, and this had to be supplemented with the actual, raw activations that the neuron has when given these images to provide a much clearer picture on what the model is doing. It also revealed that CLIP doesn't look at these fine-grained details; it's more focused on the overall composition and semantics of the image, and it also ignored things like color.

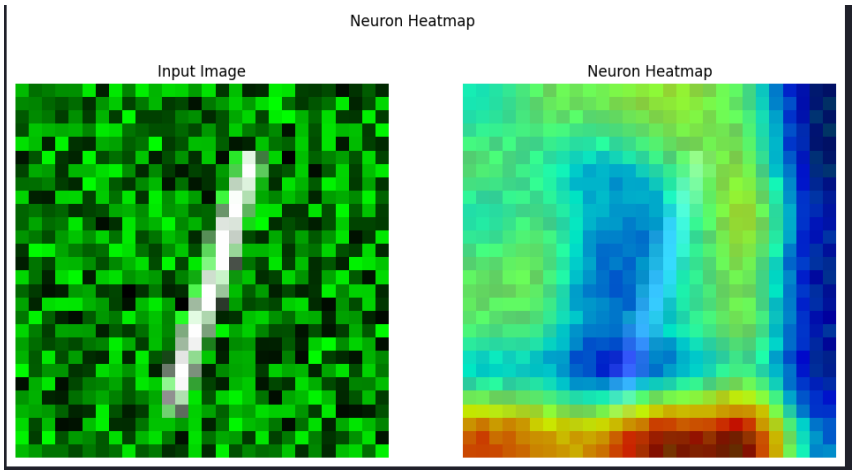

Figure 9: Example of the visual snippet (C3 Based Heatmap)

- The next thing I did was to use the activations of the c3 layer and project that onto the input for each neuron (in the notebook I have done it for neuron 2), and try to gain some insight into what the model is looking at through that. I found that it largely ignores the shape and focuses more on the background (which is in-line with

the previous findings).

- Expanding on this, and also using the 'visual snippet' + neuron activation idea from the previous point, I triednmodifying a specific neuron's activation during an inference to see what would happen. In this case, since the 2nd neuron's top images contained a lot of 1s, I:
  - saw what the predicted class was without modifications, given it was 0's color with the shape of 1. It gave a prediction of 0 (as expected).
  - saw what the predicted class was when I injected the activation that the 2nd neuron would have had if the background was the green-ish texture that activates it the most. In this case, the predicted class was still 0, and the patched prediction had increased.
  - saw what the predicted class was without modifications given the color of 4 and the shape of 1. It gave a prediction of 4 (as expected).
  - saw what the predicted class was when I injected the activation that the 2nd neuron would have had if the background was the green-ish texture that activates it the most. In this case, the predicted class was still 4, and the patched prediction had decreased.

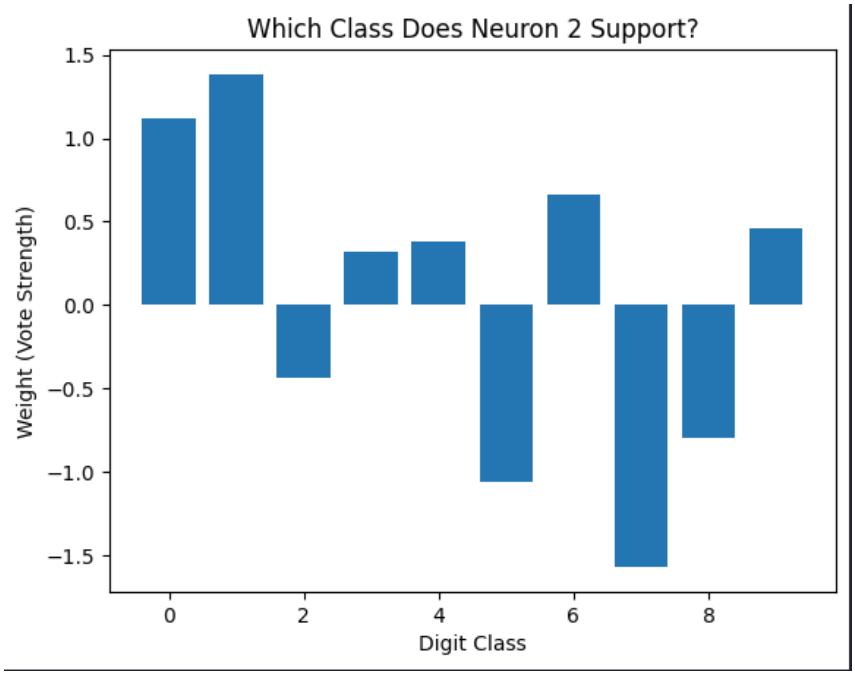

Figure 10: How neuron 2's activation is weighted

- These findings were then explained when I inspected the linear layer of the model and saw how the activation of neuron 2 is weighted in the final prediction. Although neuron 2 supports 1, it also supports 0 and 4. The reason why 0's prediction increased is because it has a high weight (> 1) for 0. The reason why 4's prediction decreased is because it has a low weight (< 1) for 4, and also the probability of some other class most likely increased a lot (since 2 is an 'influential' neuron, and silencing it causes the most change in the final prediction).

Conclusion for Neuron 2:

- Since I was majorly analyzing neuron 2, the conclusion I came to was that this my

findings kind of explain the correlation between 2 and 8, and also 2 and 13 (from the NMI graph); 2 activates 6 as well, and there were a lot of 6s seen in the top activating images for 8 and 13. So this is in-line with our previous observations.

- The conclusion for neuron 2 specifically is that it looks at both shape and texture (so it is polysemantic). It gets highly activated by a single closed loop (as seen by 0, 6 and 9) but the greatest activation happens when it sees the color 1's background texture. Also, it doesn't really like the color blue (seeing how the activation decreased when blue was introduced).

4. **Automated Process (General Application)**:
   - Next, as an attempt to automate this process, I decided to just simply run the 'activation of neuron with anti-color & digit' vs. 'activation with textured background' that we did for neuron 2, for all the neurons as a general thing that could be applied.
   - I did this by defining the anti colors for each of the digits (for a given RGB, I'd take (1-R)(1-G)(1-B) as the anti-color given normalized values).
   - I then ran this for all the neurons and found that a lot of the neurons were very texture-heavy, and only a small few were shape-heavy.
   - So, I wanted to see how removing these 'texture-heavy' neurons (who also happened to have high accuracy-deciding contributions found from the neuron ablation) would affect the model's performance.
   - I found that removing the texture-heavy neurons increased the model's performance on the hard test set (it went from ∼3% to ∼14%) and removing shape-heavy neurons then reduced the model's performance on the hard test set (it went from ∼14% to ∼10%). It was cool to see this being in line with what should happen theoretically.

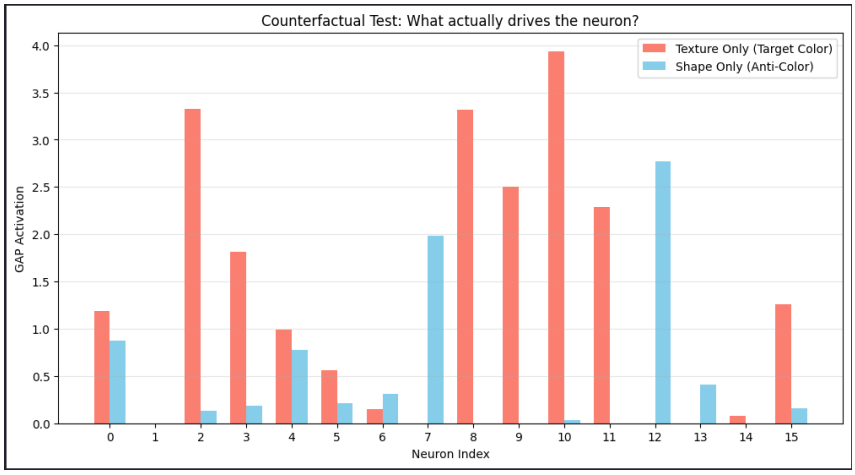

Figure 11: Example of the visual snippet (What drives each neuron)

4. Task 3: Intervention & Localization

- **Grad-CAM Visualization**:
  - I found that the heatmaps actually showed the model looking at the shape in some cases, but the final decision was still made based on the background texture/color in most cases.
  - I have implemented Grad-CAM based on the c3 layer.

– I have used the formula: $L^c_{Grad-CAM} = ReLU(\sum_k \alpha^c_k A^k)$

Figure 12: Grad-CAM

5. Task 4: Mitigation Strategies

For this, I tried several different methods, but then settled on the following two:

- **Method 1: Just Train Twice (JTT)**:
  - **Explanation**: We trained an initial "identification" model for a few epochs to identify "hard" examples (those it misclassified). We then trained a final model upweighting these error samples by a factor of 20 using a `WeightedRandomSampler`. I initially chose an upweighting factor of 20 because of the knowledge that $\sim 5\%$ of the dataset is the data that I specifically want the model to focus on. It used 20 epochs to get to this point (it had passed 70% accuracy by the 13th epoch). It uses the SGD optimizer. The learning rate, batch sizes and number of epochs were decided through trial and error.
  - **Result**: **87%** Accuracy on the Hard Test Set.
- **Method 2: K-Means Clustering**:
  - **Explanation**: I extracted features from the GAP layer and clustered them per class using K-Means (k=2). I identified the smaller cluster as the "bias-conflicting" group and upweighted it during training. For the training loop, I also used a `WeightedRandomSampler` to sample from the data, further improving the model's ability to learn shape. It uses the SGD optimizer. Again, the learning rate, batch sizes and number of epochs were decided through trial and error.
  - **Result**: **87%** Accuracy on the Hard Test Set.
- **Comparison**:
  - JTT and K-Means Clustering proved equally effective (87% vs 87%) but JTT requires training two models, and also required more epochs. While both methods utilize weighted sampling, the key difference lies in how the weights are derived (supervised error signal vs. unsupervised clustering).

- Both methods significantly outperformed the baseline (3.77%), demonstrating that the model *can* learn shape if the spurious correlation is de-emphasized.
- Here both ideas are essentially to make the model focus more on the few 'correct' examples in the dataset.
- I had a few more ideas for solving this, however I was unable to implement them fully:
  * Another idea I had was to add a color head to the model, which would use the color labels that I have from when I generated the dataset to learn to predict the color of the digit. Then, I could redefine the loss function of the model to be something along the lines of $L = L_{digit} + \lambda D_{KL}(P_{color}||U_{uniform})$
  * If we have a measure of 'how complex' as model is (like the local learning coefficient), then we should be able to measure the complexity of a model that we have (for example, the baseline that we have) and then use it's measured complexity as an indicator of if the model that we are currently training is getting 'too simple'. For example, in our case, we'd measure the LLC of the baseline, and then during the training of the robust model we'd continually measure the LLC epoch and if the model's LLC is getting to low, we could have an 'intervention' and try to make the model intentionally step out of this minima.

6. Task 5: Adversarial Attacks

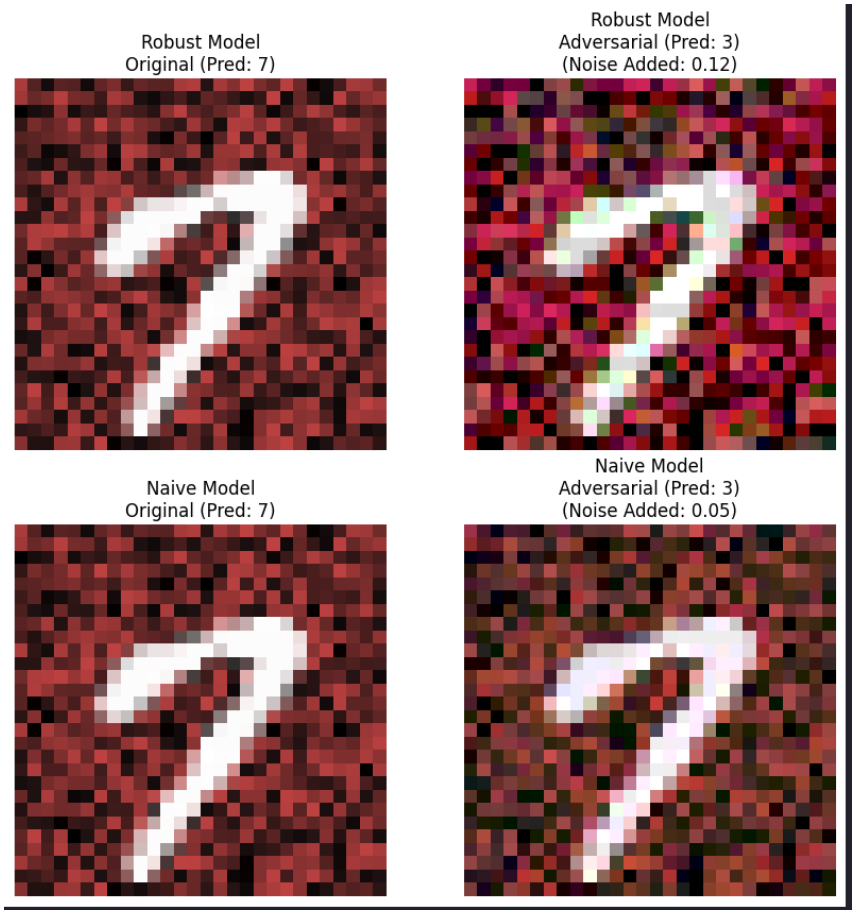

Figure 13: Adversarial Attacks

- For this, I used Projected Gradient Descent (PGD).

- **The robust model required a noise magnitude of $\sim$ 0.12 to be fooled.**
- **The naive model required a noise magnitude of $\sim$ 0.05 to be fooled.**
- I used the gradient's sign to normalize the step size, and used clamp as the projection to ensure the noise stayed within the bounds defined.

7. Extra Learnings: Complexity & Singular Learning Theory (SLT)

Over the course of the task I came across SLT and I thought it was pretty interesting; here's what I learnt.

Theoretical Explanation

Singularities and the Geometry of Loss

We can express a model as a probability distribution $p(y|x, \theta)$ where θ represents the parameters of the model and x represents the input. We assume there exists an ideal, "true" probability distribution $q(y|x)$ which depends only on the input x. Analogously, q represents the "true labeller"; the actual probability of x being assigned the label y by the human annotator.

We can define the loss function $K(\theta)$ as the KL divergence between the true distribution q and the model p:

$$K(\theta) = KL(q||p)$$

This measures the information lost when approximating the truth q with the model p.

We define the **Parameter Space** $W = \mathbb{R}^d$ as the set of all possible weights and biases. The neural network acts as a mapping from this parameter space to the space of probability distributions. Now, imagine a $\mathbb{R}^{d+1}$ space where the added dimension represents the loss function $K(\theta)$, which is a scalar quantity dependent on θ (an easy way to imagine this is to set $d = 2$. If θ_1 and θ_2 are the components of our 2-dimensional parameters (let's say they're on the x and y axis), then the z axis represents the loss function $K(\theta)$ which can be determined given θ). This is called the **Loss Landscape.** So now we can see each θ as an identifier for a specific trained model, where $K(\theta)$ represents its 'distance' from the true distribution q.

A model is called **regular** if the mapping from parameters θ to probability distributions $p(x|\theta)$ is one-to-one (identifiable) and given a point, the loss surface *looks like a bowl* in an infinitesimal neighborhood around that point. Specifically, identifiability means that if $p(x|\theta_1) = p(x|\theta_2)$, then $\theta_1 = \theta_2$.

However, neural networks are fundamentally **singular**. This means that if $p(x|\theta_1) = p(x|\theta_2)$, then we can not necessarily say that $\theta_1 = \theta_2$, and also that the loss surface does not necessarily look like a bowl in an infinitesimal neighborhood around a given point. We know that neural networks are singular because many different parameter settings represent the *function*. Permuting hidden units, scaling one layer up and the next down, and so on collapse distinct parameter points onto the same model. So, we have essentially gotten different weights for the same probability distribution, which means we have a singular model (through non-identifiability). This is the basis for **Singular Learning Theory.**

From non-identifiability, we could have several θ such that $p(\cdot|\theta) = q(\cdot)$. We call the set of all such θ the **zero set**, denoted by Θ_0. So, by definition:

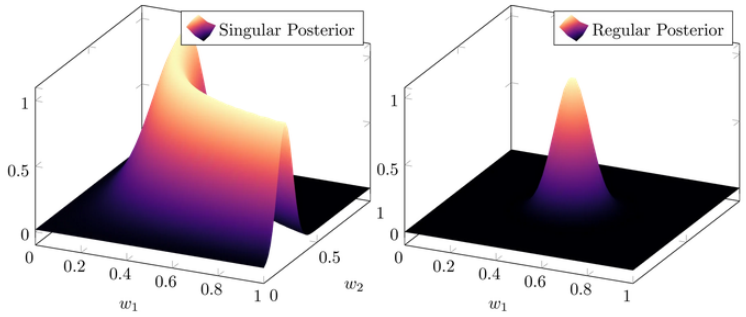

Figure 14: Regular vs Singular Models

$$\Theta_0 = \{\theta \in \mathbb{R}^d : K(\theta) = 0\} = \{\theta : p(\cdot \mid \theta) = q(\cdot)\}$$

To connect this geometric structure to learning behavior, we consider the Bayesian marginal likelihood (or evidence). Let $D = \{(x_i, y_i)\}_{i=1}^n$ be i.i.d. samples drawn from q. The likelihood is

$$p(D \mid \theta) = \prod_{i=1}^{n} p(y_i \mid x_i, \theta),$$

and we define the empirical negative log-likelihood

$$L_n(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log p(y_i \mid x_i, \theta).$$

Each term $-\log p(y_i \mid x_i, \theta)$ is a random variable with respect to $(x_i, y_i) \sim q$. By the law of large numbers,

$$L_n(\theta) \xrightarrow[n\to\infty]{} \mathbb{E}_q[-\log p(y \mid x, \theta)]\,.$$

This expectation can be decomposed as

$$\mathbb{E}_q[-\log p(y \mid x, \theta)] = \mathbb{E}_q[-\log q(y \mid x)] + \mathrm{KL}(q \| p_\theta),$$

(from the definition of KL divergence) where the first term is independent of θ. Therefore, up to an additive constant,

$$L_n(\theta) \;\to\; K(\theta).$$

With a prior density $\varphi(\theta)$ over parameters, the marginal likelihood is

$$p(D) = \int p(D \mid \theta)\, \varphi(\theta)\, d\theta \;\approx\; \int \exp\big(-nK(\theta)\big)\, \varphi(\theta)\, d\theta.$$

We define $p(D)$ because it tells us how well the entire model class, i.e. all distributions representable by parameters $\theta \in \mathbb{R}^d$ fits the data, and not just the best fitting parameter. $p(D \mid \theta)$ represents one such instance of a model belonging to a model class.

Getting back to the integral, for large n, the exponential term $\exp(-nK(\theta))$ concentrates mass near the zero set Θ_0 and goes to zero for parameters with large loss. Define the ε-neighbourhood

$$U_\varepsilon = \{\theta : K(\theta) \le \varepsilon\}.$$

$\exp(-nK(\theta))$ can be seen as an indicator of $U_{1/n}$, so the integral above effectively measures the prior-weighted volume of a neighborhood of thickness $1/n$ around Θ_0:

$$p(D) \approx \int_{U_{1/n}} \varphi(\theta)\, d\theta.$$

In a regular model, $K(\theta)$ grows quadratically away from a unique optimum, and $\mathrm{Vol}(U_\varepsilon) \sim \varepsilon^{d/2}$. In a singular model, Θ_0 has positive dimension or intersecting components, and the loss grows more slowly in some directions. As a result, the volume scales as

$$\mathrm{Vol}(U_\varepsilon) \sim \varepsilon^\lambda, \qquad \lambda < \frac{d}{2}.$$

The exponent λ captures the geometric complexity of the zero set and is called the **real log canonical threshold** (RLCT). Formally, λ can be defined using the zeta function

$$\zeta(z) = \int K(\theta)^z\, \varphi(\theta)\, d\theta,$$

whose largest pole is at $-\lambda$ which is the asymptotic scaling of the volume above. (A pole of order k of any general function $f(z) = \frac{g(z)}{(z-z_0)^k}$ is defined as z_0 given $g(z) \neq 0$ at z_0.)

Using this scaling, the marginal likelihood satisfies

$$-\log p(D) = n \inf_\theta K(\theta) + \lambda \log n + O(1).$$

Thus, in singular models, learning behavior is governed not by the ambient parameter dimension d, but by the geometry of the zero set Θ_0 as summarized by λ.

Localization of the RLCT - the Local Learning Coefficient

The real log canonical threshold (RLCT) λ defined above is a global quantity: it is determined by the most singular region of the zero set Θ_0 and governs the asymptotics of the full marginal likelihood $p(D)$. However, the zero set Θ_0 may contain regions with different local geometries, corresponding to different degrees of singularity.

To describe this, let $U \subset \mathbb{R}^d$ be an open neighborhood in parameter space, typically centered around a point or region reached by training. We define the **localized zeta function:**

$$\zeta_U(z) = \int_U K(\theta)^z\, \varphi(\theta)\, d\theta.$$

As before, if $\lambda(U)$ denote the largest pole of $\zeta_U(z)$, then $\lambda(U)$ is called the **local real log canonical threshold** associated with the region U.

Equivalently, $\lambda(U)$ characterizes the scaling of the volume of low-loss parameters inside U. Defining the local sublevel set

$$U_\varepsilon = \{\theta \in U : K(\theta) \leq \varepsilon\},$$

we have the asymptotic relation

$$\int_{U_\varepsilon} \varphi(\theta)\, d\theta \sim \varepsilon^{\lambda(U)} \qquad (\varepsilon \to 0).$$

The **global** RLCT is recovered by minimizing over all neighborhoods:

$$\lambda = \inf_U \lambda(U).$$

In contrast, when learning dynamics or posterior concentration restrict the parameters to a particular region U, the effective learning behavior is governed by $\lambda(U)$ rather than by the global λ. This localized exponent is often referred to as the **local learning coefficient (LLC)**.

Empirical Analysis

- **Motivation**: Standard metrics (Accuracy/Loss) don't tell the full story. A simple solution (Color mapping) and complex solution (Shape recognition) might have similar Training Loss, but vastly different generalization.
- **Local Learning Coefficient (LLC)**:
  - **Definition**: The LLC estimates the 'effective dimensionality' or complexity of the function the model has learned. A higher LLC implies a more complex/structured solution.
  - **Estimation Method**: We used SGLD (Stochastic Gradient Langevin Dynamics) to estimate the RLCT (Real Log Canonical Threshold).
- **Results**:
  - **Baseline Model LLC**: $\sim$**1.5**
  - **Robust (Clustered) Model LLC**: $\sim$**2.5**
- **Interpretation**:
  - The baseline model found a 'simple' solution (color mapping), resulting in a lower complexity score ($\sim$1.5).
  - The robust model was forced to learn 'shape', a more complex feature, resulting in a significantly higher LLC ($\sim$2.5).
  - This quantitatively validates that the "robust" solution is distinct and simpler solutions (like color bias) are preferred by SGD unless intervention occurs.

8. Conclusion

- **Summary**: We successfully demonstrated that standard training on biased data leads to spurious correlation learning (3.77% accuracy). Interpretability tools (Grad-CAM, Feature Viz) confirmed the model was a "color detector". Mitigation strategies like JTT and Unsupervised Clustering recovered robust accuracy (up to 85.7%).
- **Trade-off**: The analysis highlights a fundamental trade-off: models prefer simple, spurious solutions. Achieving robustness requires active intervention (re-weighting data) to force the model to learn the more complex, generalized features (shape), as confirmed by SLT complexity metrics.