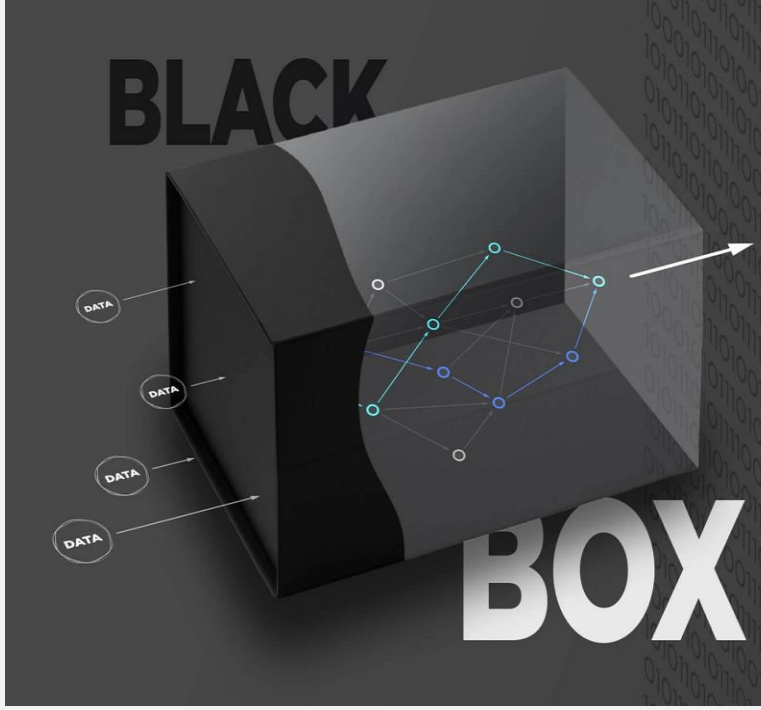
UNBOXING-BLACKBOX



THE LAZY ARTIST-CV

Mātaram Pitrum Vande|
Āchārya Devo Bhava

By P. Srivatsav Reddy
2025121016

TASK0- The Biased Canvas:

I made background textured and colored it
and foreground strokes gray to prob insane !

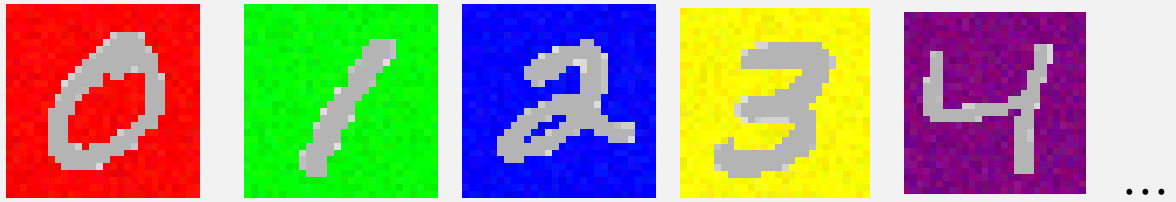
EASY TRAIN

Digit	Dominant Color Count	Total	Percent
0	5615	5923	94.80%
1	6409	6742	95.06%
2	5692	5958	95.54%
3	5835	6131	95.17%
4	5521	5842	94.51%
5	5138	5421	94.78%
6	5642	5918	95.34%
7	5959	6265	95.12%
8	5565	5851	95.11%
9	5657	5949	95.09%

HARD TEST

Digit	Dominant Color Count	Total	Percent
0	0	980	0.00%
1	0	1135	0.00%
2	0	1032	0.00%
3	0	1010	0.00%
4	0	982	0.00%
5	0	892	0.00%
6	0	958	0.00%
7	0	1028	0.00%
8	0	974	0.00%
9	0	1009	0.00%

Train set: 95 percent each-digit:biased towards same colour



Test set: negation of colors

[Created kaagle datacard : ttps://www.kaggle.com/datasets/poreddysr/c2ty2bcouloredspurious](ttps://www.kaggle.com/datasets/poreddysr/c2ty2bcouloredspurious)

TASK 1- The Cheater:

Before cheating , I used proper STRATIFIED Split in the dataset and used 25percent of data MNIST Training ALL models out of 60k training 10k testing i.e, 12k training 3 val 2.5 test

```
 Easy Train counts: {0: 1185, 1: 1349, 2: 1191, 3: 1226, 4: 1169, 5: 108
4, 6: 1183, 7: 1253, 8: 1170, 9: 1190}
 Easy Val counts  : {0: 296, 1: 337, 2: 298, 3: 307, 4: 292, 5: 271, 6: 2
96, 7: 313, 8: 293, 9: 297}
 Hard Test counts : {0: 245, 1: 284, 2: 258, 3: 253, 4: 245, 5: 223, 6: 2
40, 7: 257, 8: 243, 9: 252}
```

For fast Training results I used GPU-T4 in kaagle, NUM WORKERS=2 and PIN MEMORY = True in batch loaders in PYTORCH framework

SIMPLE CNN:

```python
class LazyCNN(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3,6, kernel_size=9, stride=2, padding=4)   # 28 → 14
        self.conv2 = nn.Conv2d(6, 8, kernel_size=7, stride=2, padding=3)   # 14 → 7
        self.conv3 = nn.Conv2d(8, 16, kernel_size=3, stride=1, padding=1)  # 7 → 7

        self.fc = nn.Linear(16 * 7 * 7, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = torch.flatten(x, 1)
        return self.fc(x)
```

```python
resnet_transform = transforms.Compose([
    transforms.Resize(256, interpolation=InterpolationMode.BILINEAR),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

```python
transform = transforms.ToTensor()
```

Training Accuracy:95.15%
Easy Validation Accuracy: 95.13%
HARD test accuracy :7.04%

RESNET 18:

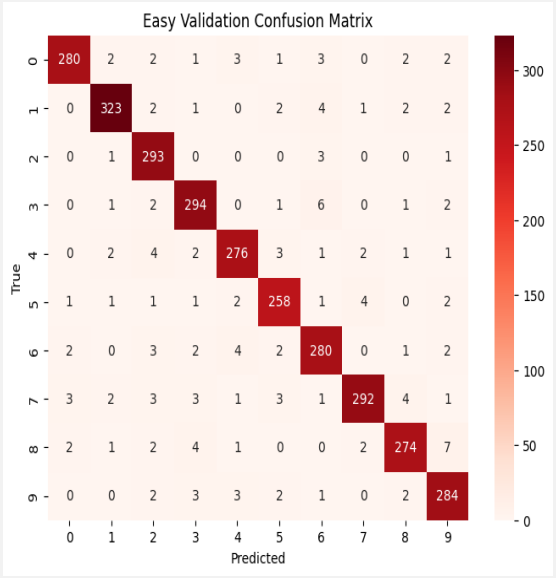
Ya I didn't give up when my first attempt was hard test accuracy:90
without improper data transformations according to pytorch documentation and I didn't unfreeze the models conv's

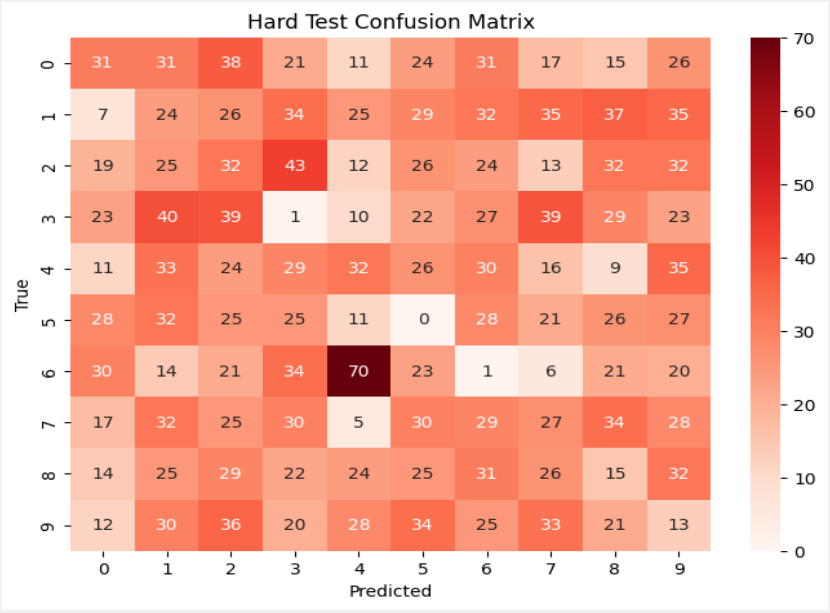
When unfreezed 2 layers models test accuracy was 0 percentage yes even HUGE BRAINS ARE biased

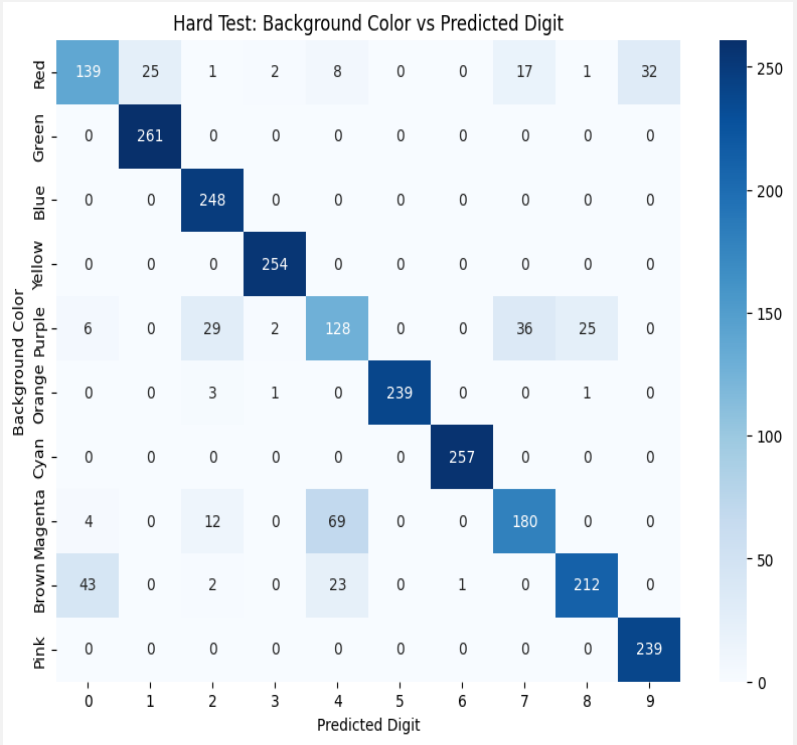
When unfreezed only layer4+fc only:

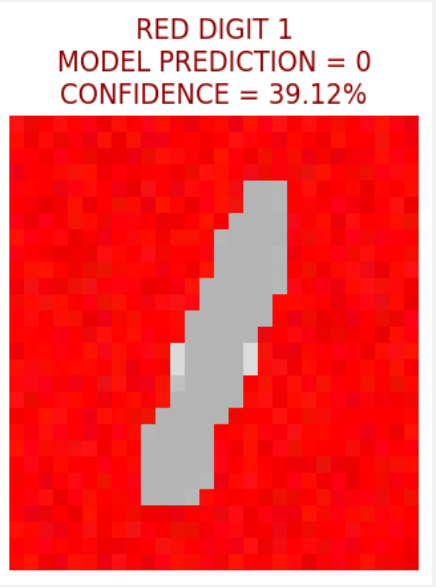
Training Accuracy:95.36%
Easy Validation Accuracy: 95.23%
HARD test accuracy :7.56%

SIMPLE CNN:

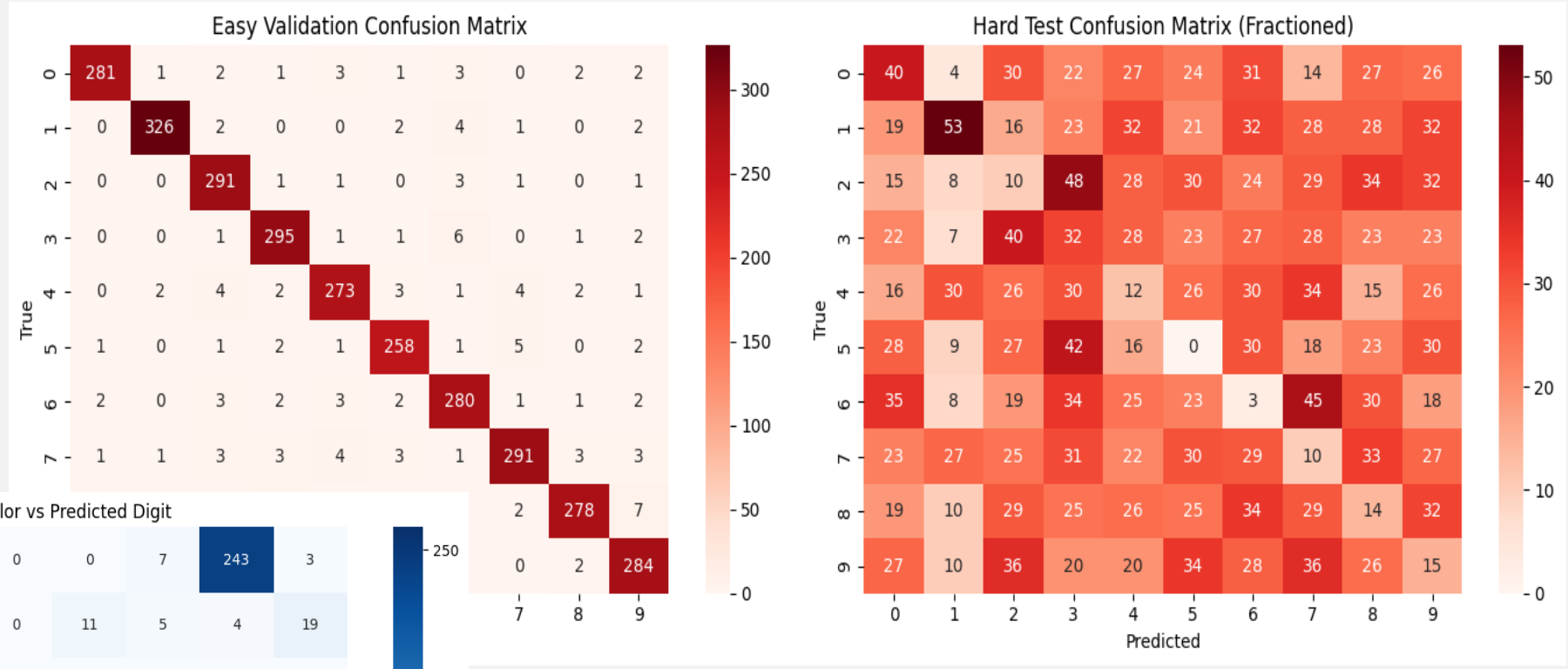

Easy Validation Confusion Matrix
True
Predicted

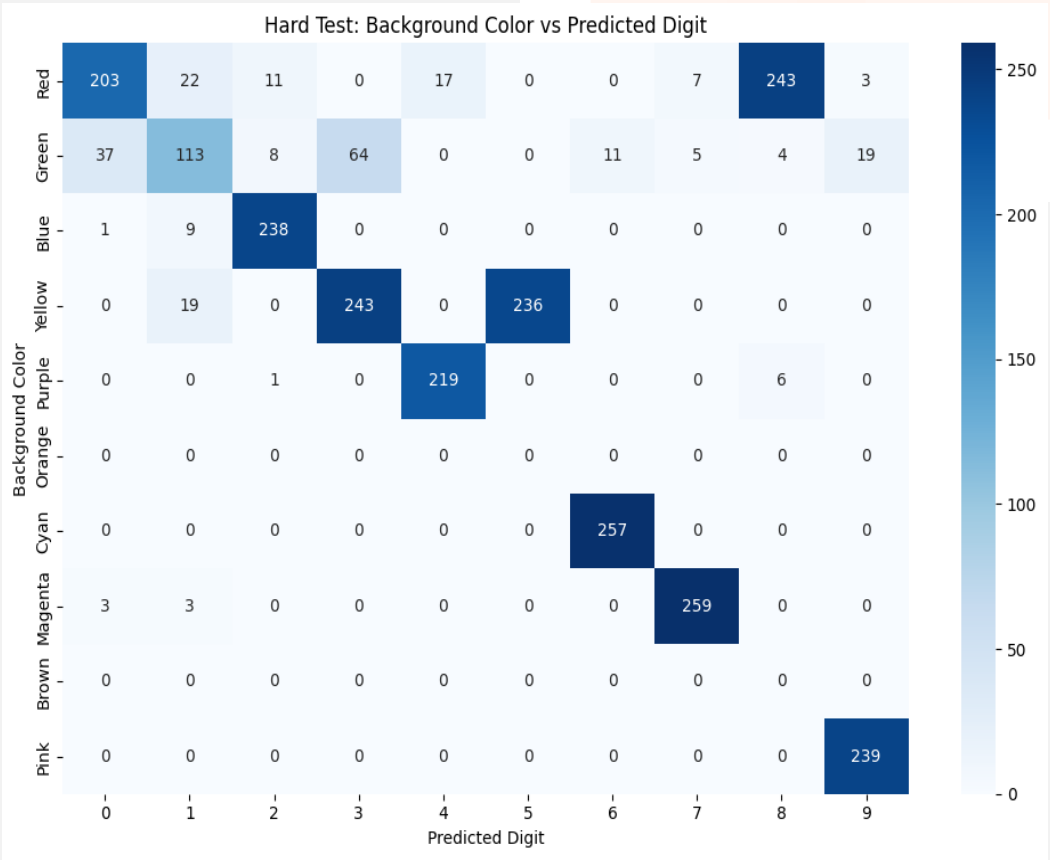

Hard Test Confusion Matrix
True
Predicted


Hard Test: Background Color vs Predicted Digit
Background Color
Red
Green
Blue
Yellow
Purple
Orange
Cyan
Magenta
Brown
Pink
Predicted Digit


RED DIGIT 1
MODEL PREDICTION = 0
CONFIDENCE = 39.12%

RESNET 18:

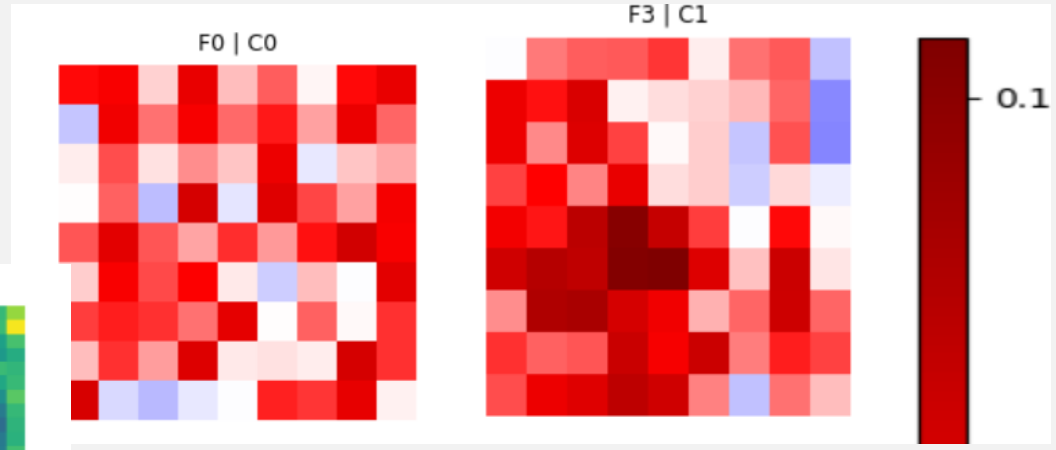




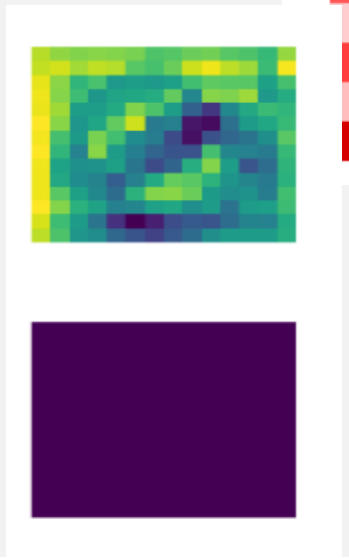
SEEMS LIKE RESNET IS EVEN BIASED : red is splitted because brown is kinda reddish brown

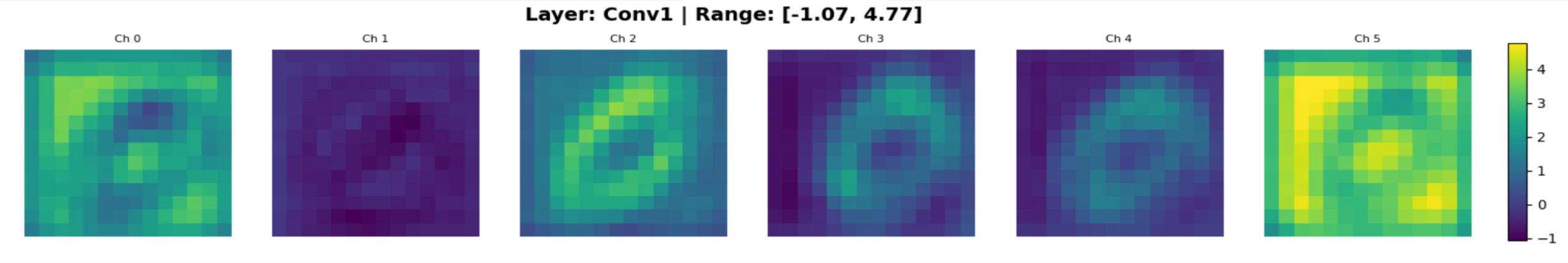
Task 2- The Prober:

These are simple 3LCNN
FILTERS MOST WEIGHTS ARE FOCUSED ON BACKGROUND:

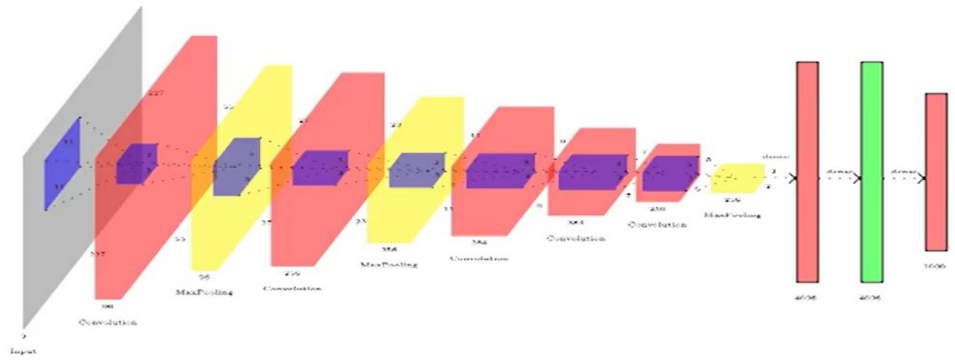


I ensured proper scaling is done for feature maps :





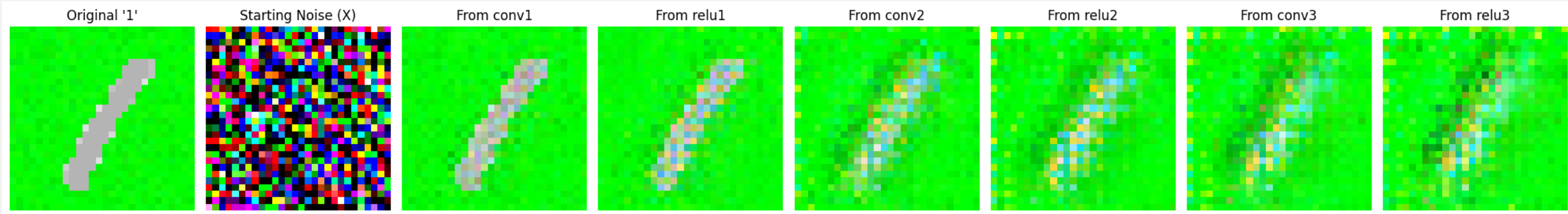
I created image embededdings of 3 Lsimple cnn:



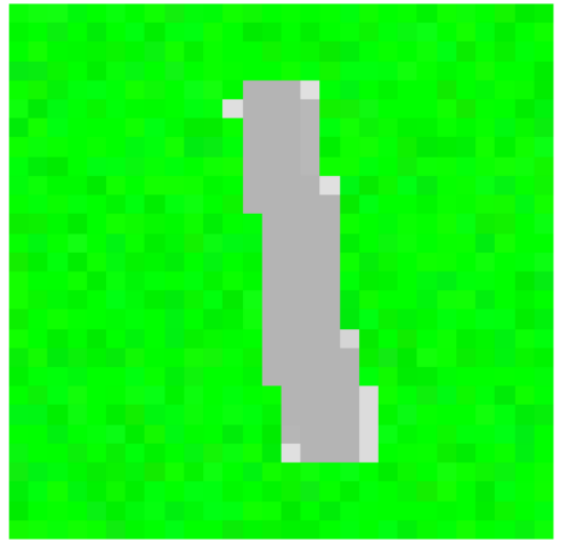
- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)
- Repeat
  - Forward pass using X and compute $\phi(x)$.
  - Compute

$$\mathscr{L}(i) = ||\phi(x) - \phi_0||^2 + \lambda||\phi(x)||_6^6$$

  - $i_k = i_k - \eta\frac{\mathscr{L}(i)}{\partial i_k}$



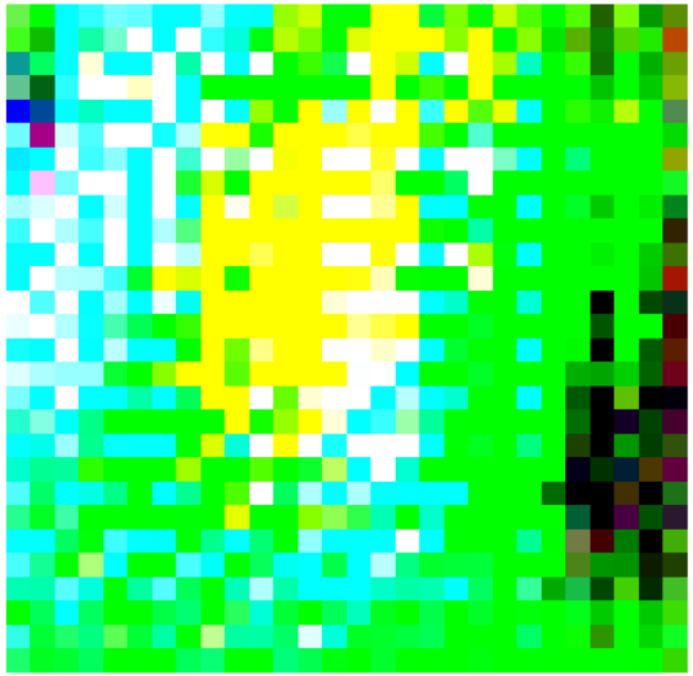
I deep dreamed :



```
--- Step 2: Precomputing 50 iterations ---
100%|██████████| 50/50 [00:00<00:00, 157.17it/s]
```

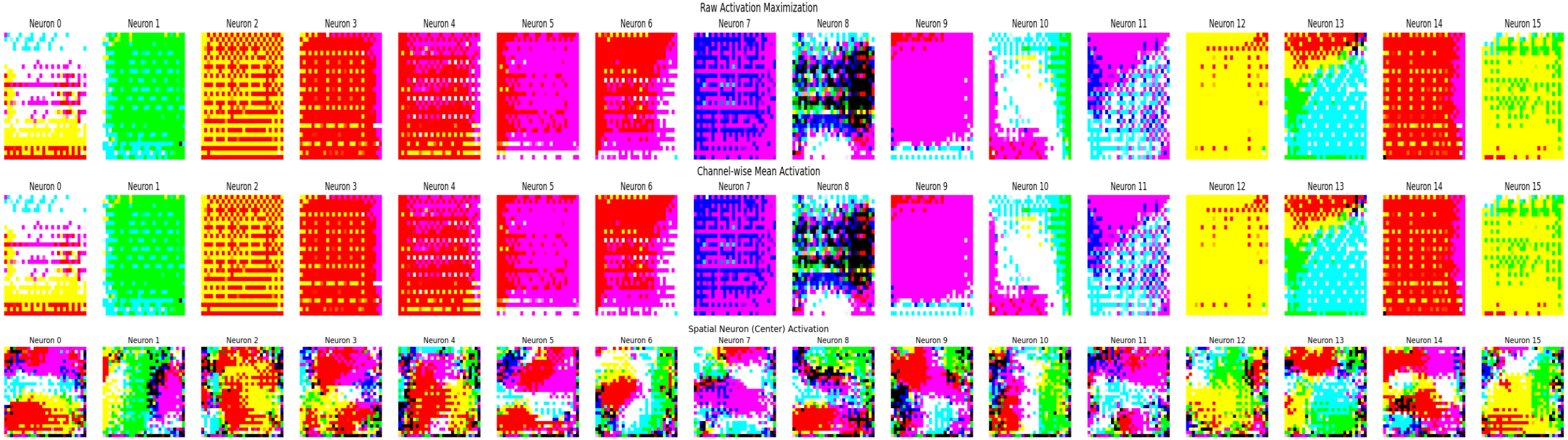
Iteration: 50

Iteration: 50 | Probing: Target Class 1

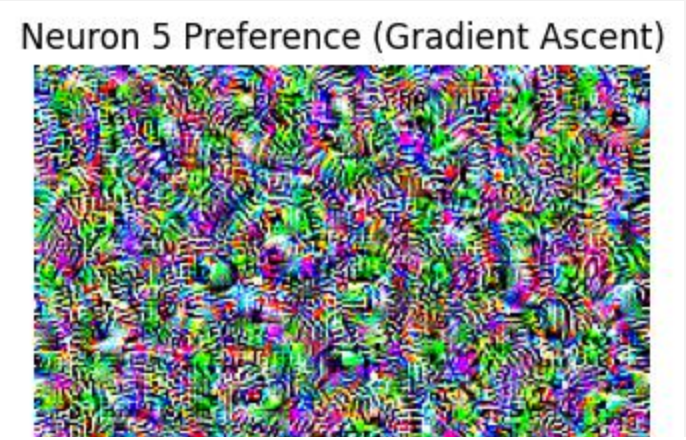


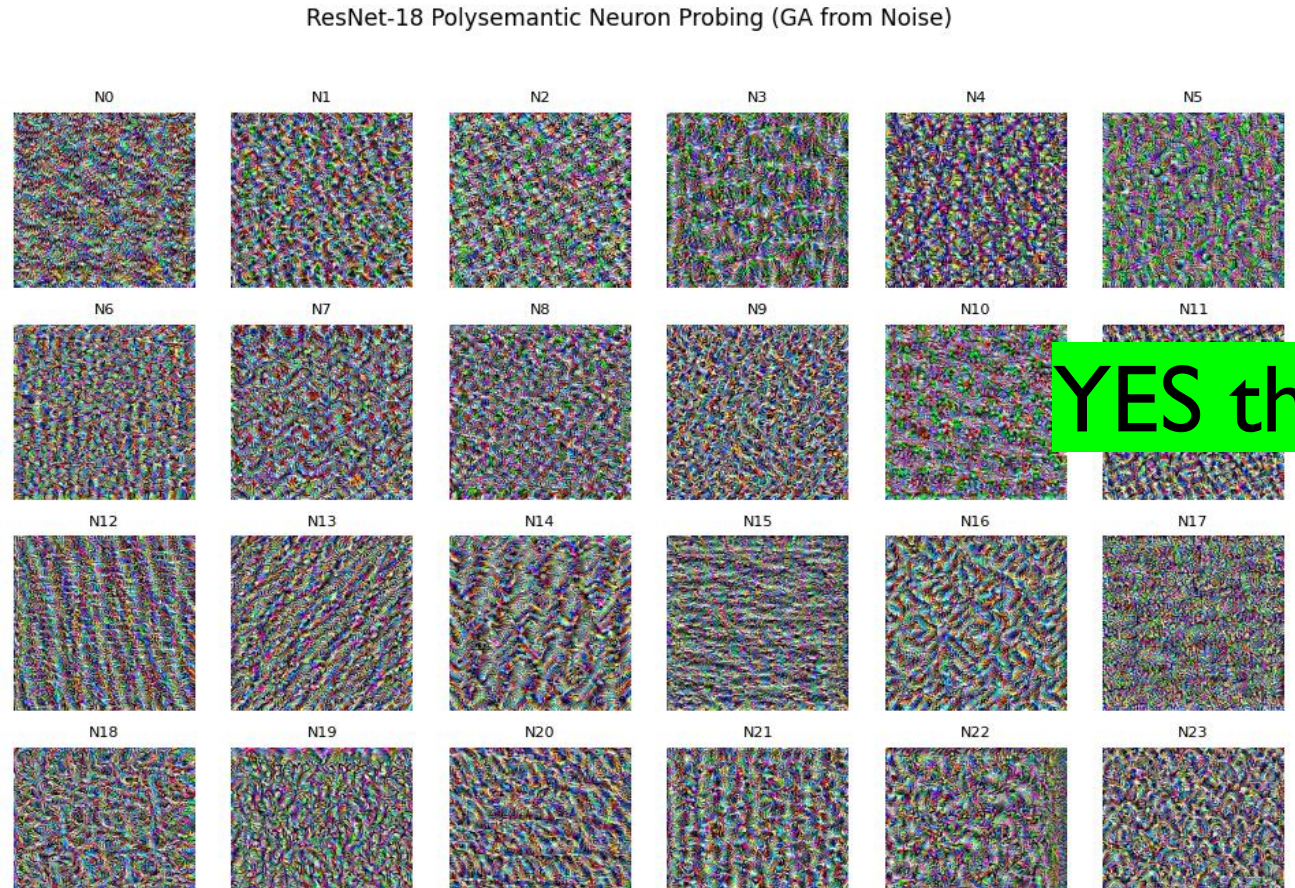
Finally! from optimizable image tensor to : raw|channelwise mean|spatial activation mean activations @ conv3 for simple3LCNN:

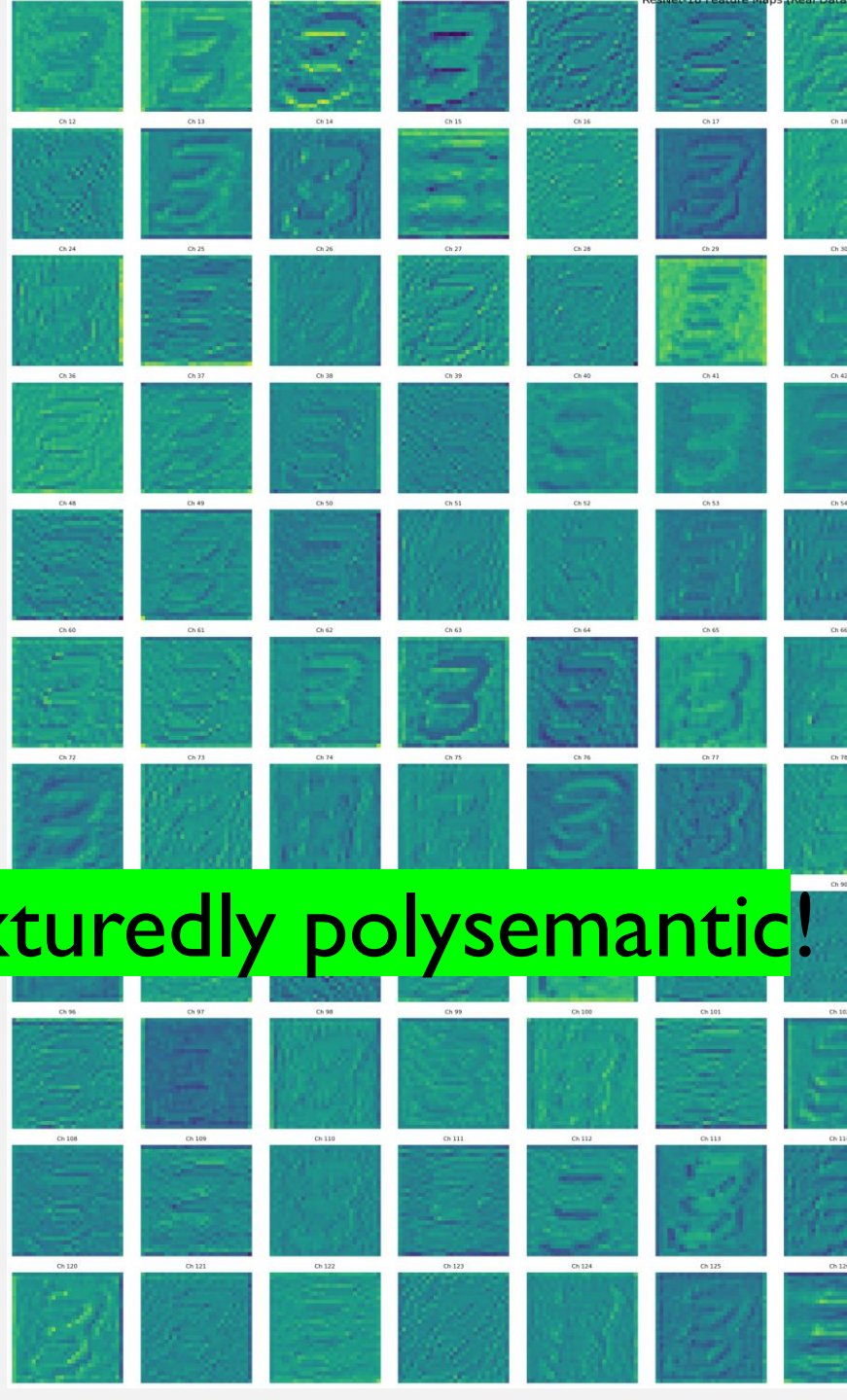
Let's see these ending in task 6! YES THEY ARE COLOURED ,TEXTUREDly activated ARE THEY HAVING DIGIT AVTIVATIONS EVEN?????:



FROM RESNET18:



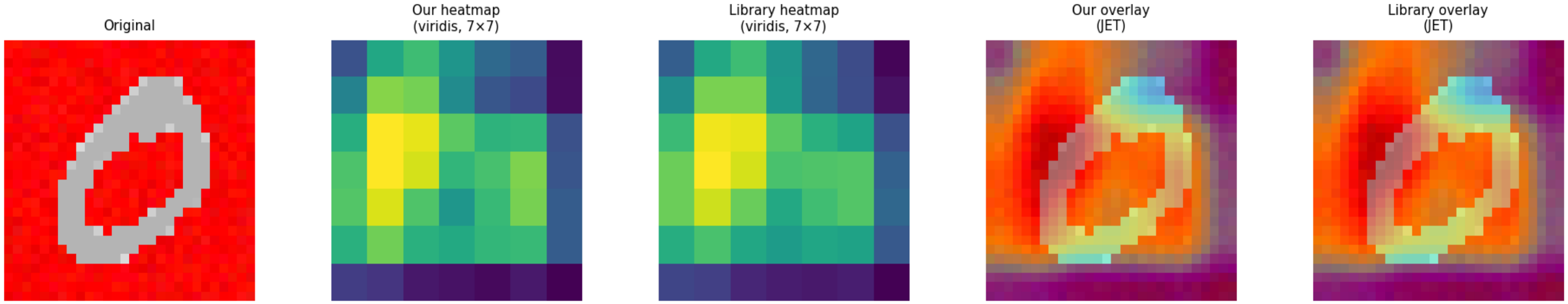




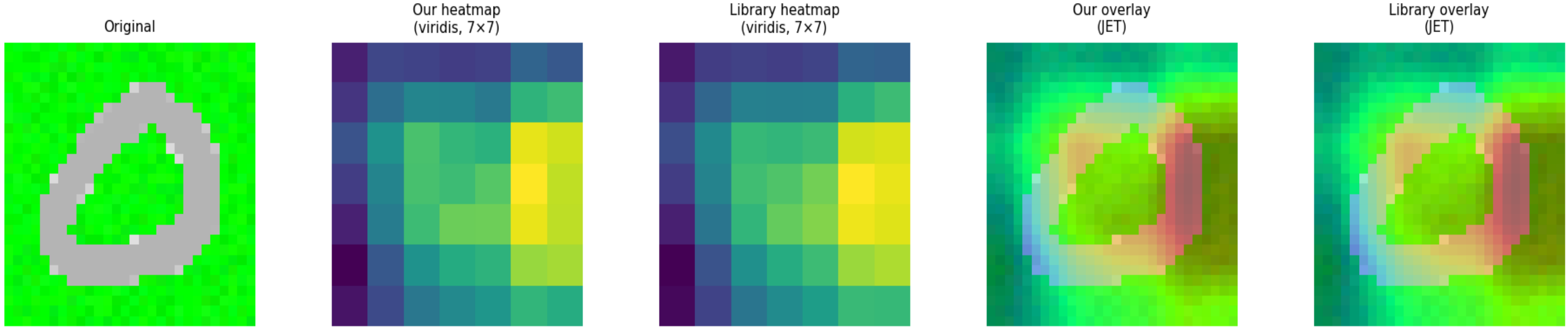
YES they are texturedly polysemantic!

MADE GRAD CAM!!!!!-interpolation problem with resnet18:

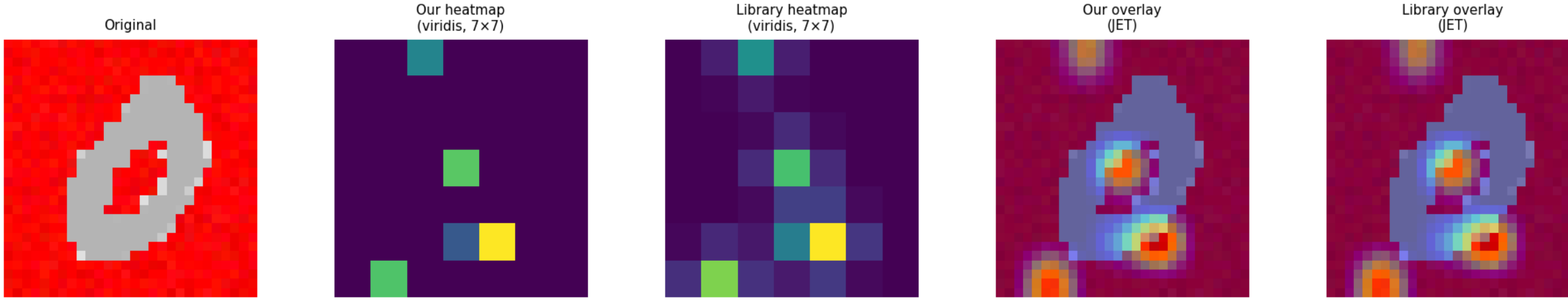
LazyCNN — Biased example: Red 0 (easy/train)
0_00001.png | pred=0 | target=0 | heatmap MAE=0.033



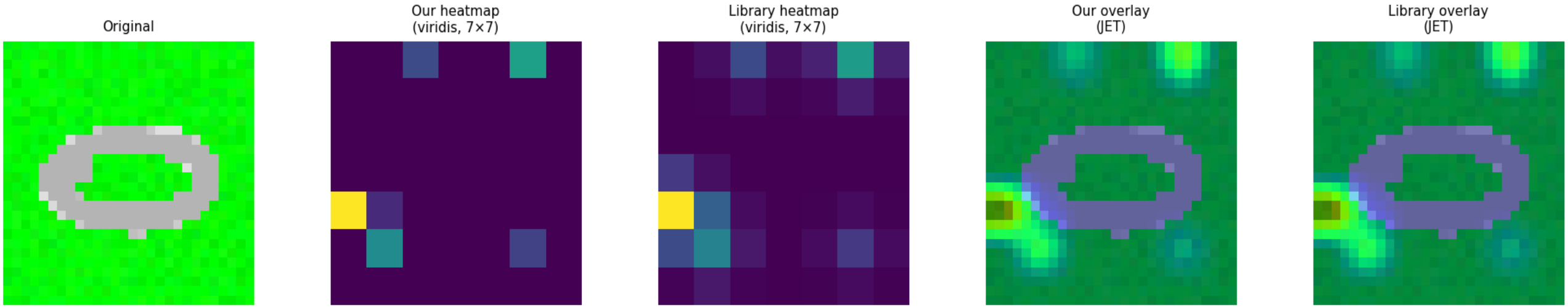
LazyCNN — Conflicting example: Green 0 (hard/test)
0_00296.png | pred=1 | target=1 | heatmap MAE=0.021



ResNet18 — Biased example: Red 0 (easy/train)
0_00037.png | pred=5 | target=5 | heatmap MAE=0.047



ResNet18 — Conflicting example: Green 0 (hard/test)
0_01191.png | pred=3 | target=3 | heatmap MAE=0.029



Task 4 - The Intervention :

My hypo'sMethod 1: USED BATCH NORM for translational invariance
:everything was same transformation…….

| Train Acc: 99.38% | Val Acc: 98.27%

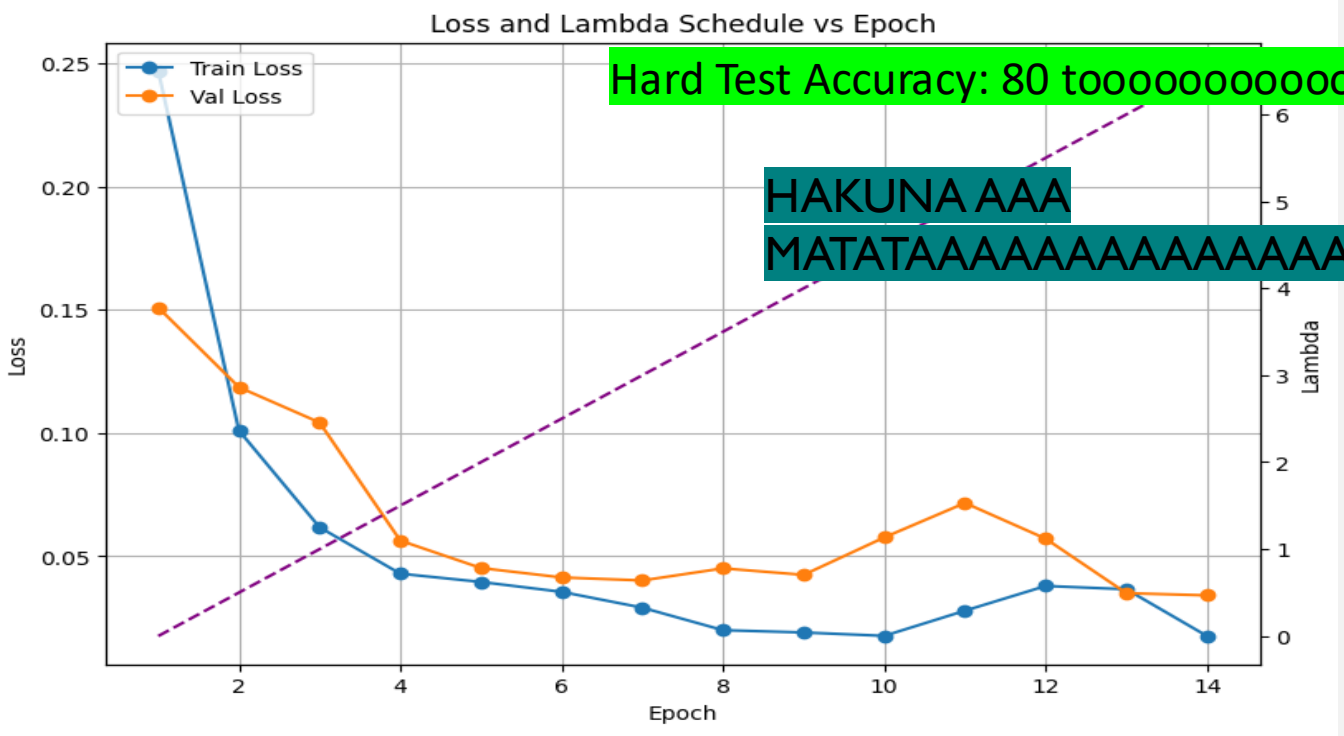
Hard Test Accuracy: 80.84

HAKUNA MATATA!!!!!

```python
import torch.nn.functional as F

class RobustCNN(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, 5, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2)  # 28 -> 14
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2)  # 14 -> 7
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU()
        )

        self.fc = nn.Linear(128 * 7 * 7, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = torch.flatten(x, 1)
        return self.fc(x)
```

Method 2: I USED BATCH NORM + COLOUR PENALISATION with OWN SCHEDULER like lr scheduler for translational invariance and colour invariance
:everything was same transformation .....

```python
def lambda_schedule(epoch):
    return LAMBDA_MAX * (epoch / EPOCHS)
```

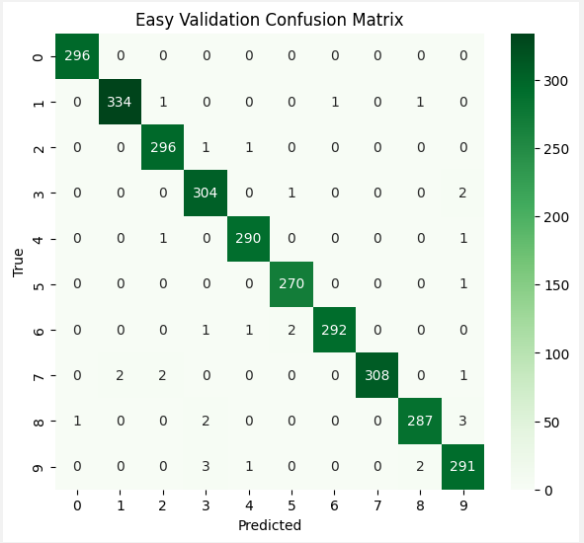


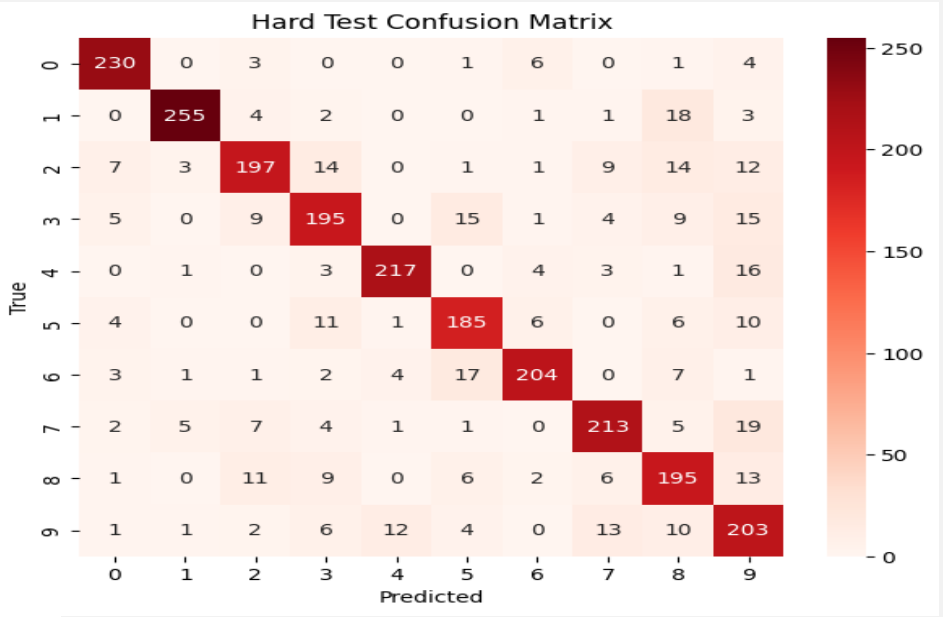
Hard Test Accuracy: 80 tooooooooooo 83.76%

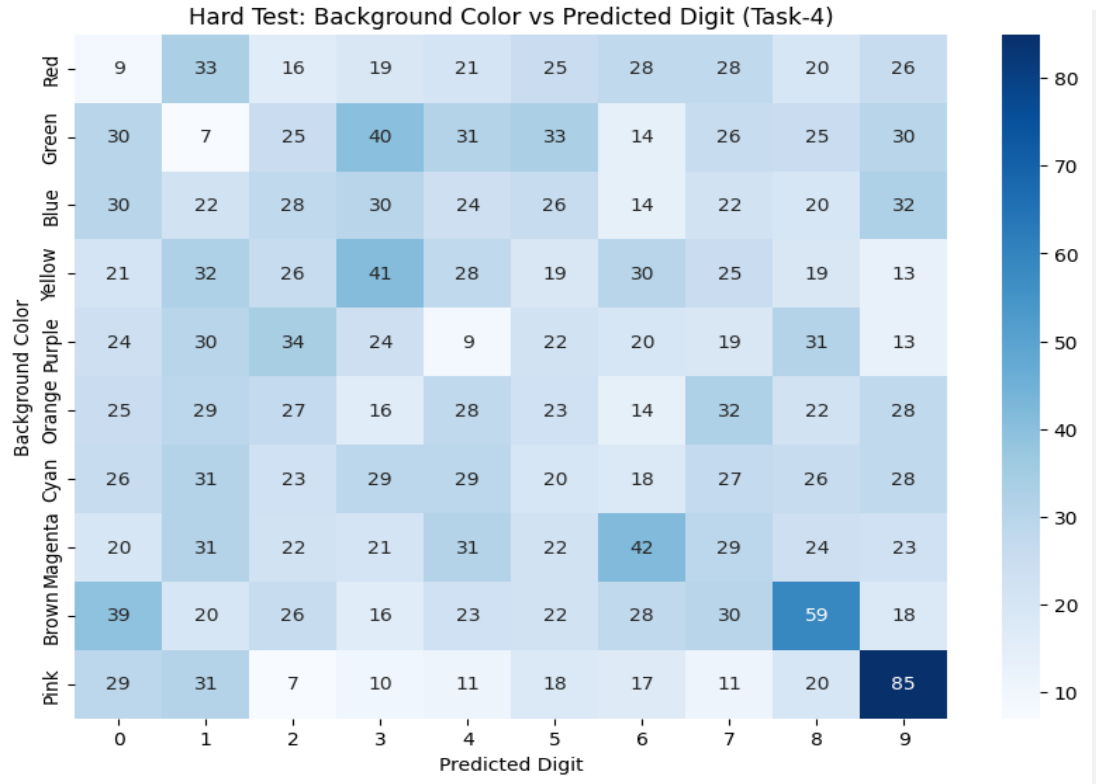
HAKUNA AAA MATATAAAAAAAAAAAAAA!!!!!

```python
def color_gradient_penalty(images, logits, labels):
    ce_loss = criterion(logits, labels)

    grads = torch.autograd.grad(
        outputs=logits.gather(1, labels.view(-1, 1)).sum(),
        inputs=images,
        create_graph=True
    )[0]

    grad_r = grads[:, 0]
    grad_g = grads[:, 1]
    grad_b = grads[:, 2]

    color_penalty = torch.var(
        torch.stack([grad_r, grad_g, grad_b], dim=1),
        dim=1
    ).mean()

    return ce_loss, color_penalty
```

TRAIN Acc: 99.79% | Val Acc: 98.93%








```
===== HARD TEST: DIGIT 1 =====
Total digit-1 samples : 284
Correct predictions   : 255
Accuracy              : 89.79%
```

Task 5: The Invisible Cloak : FIRSTLY I was dumb to attack I tried OPTUNA lib –for selection of HYPERPARAMETER's of model uses basyeain optimization in search space,its generalized used in model training ,can be used in Quant task but here I bruteforced to select perturbation of which NOISE : 7 is predicted as 3

No use which I thought for ,I thought it will endup in UNIVERSAL ATTACK like:

The thought is start with noise such that
Noise image := difference of (7 predicted as most confident
3 means there is something making the model as 3
And least confident 7 and 2nd logit as 3) started
That noise and moved to a good noise with optuna for lazy model so that we might get noise such that
Every image 7 predicted as 3!

Yeah failed because model is lazy such that its understood what to focus on even –colour pixel by pixel !

```
x7 = get_clean_7(robust_model, hard_loader)

study = optuna.create_study(direction="maximize")
study.optimize(
    lambda trial: optuna_noise_objective(trial, robust_model, x7),
    n_trials=30  # keep small, this is expensive
)

print("Best confidence for class 3:", study.best_value)
```

```
[I 2026-02-03 08:00:40,151] A new study created in memory with name: no-name-c854384c-828c-475c-ae48-8e947dc5fc81
[I 2026-02-03 08:00:40,510] Trial 0 finished with value: 8.9540253611630 76e-07 and parameters: {'d_0_0_0': -0.006588092390444847, 'd_0_0_1': 0.017820663535946912, 'd_0_0_2': 0.007576114907914287,
'd_0_0_3': -0.038181010061627274, 'd_0_0_4': 0.006504265016708963, 'd_0_0_5': 0.023521812198807196, 'd_0_0_6': -0.0446697436255481, 'd_0_0_7': 0.027499579655146628, 'd_0_0_8': -0.0310700268056090
3, 'd_0_0_9': -0.017593068687640226, 'd_0_0_10': 0.044264936554160667, 'd_0_0_11': -0.03923716117543733, 'd_0_0_12': -0.02490594024638213, 'd_0_0_13': -0.037772130992795784, 'd_0_0_14': -0.048553
117797209014, 'd_0_0_15': 0.04039406694786561, 'd_0_0_16': 0.013801237129546257, 'd_0_0_17': 4.526836480909374e-05, 'd_0_0_18': -0.037579470334260 19, 'd_0_0_19': -0.030987499431264054, 'd_0_0_2
0': -0.03304154335 13798, 'd_0_0_21': 0.039906920135984306, 'd_0_0_22': 0.001155547115512856, 'd_0_0_23': 0.021240804260968263, 'd_0_0_24': 0.0030990824435015857, 'd_0_0_25': -0.04085899769015843,
'd_0_0_26': 0.04895600520273734, 'd_0_0_27': 0.03650034669749162, 'd_0_1_0': -0.021753804214810404, 'd_0_1_1': 0.000705774813127 7936, 'd_0_1_2': 0.0023153529451426097, 'd_0_1_3': -0.0110556393459
19535, 'd_0_1_4': -0.010517936146262008, 'd_0_1_5': 0.017615763952363778, 'd_0_1_6': -0.034324469728518756, 'd_0_1_7': 0.028641351835942158, 'd_0_1_8': -0.025557474084808674, 'd_0_1_9': -0.001984
357311278148, 'd_0_1_10': -0.021320284392984015, 'd_0_1_11': -0.014499506416415633, 'd_0_1_12': -0.020458528482044627, 'd_0_1_13': 0.021357883141971234, 'd_0_1_14': 0.01318325214696936, 'd_0_1_1
5': -0.043610781699486 54, 'd_0_1_16': 0.016850975195384615, 'd_0_1_17': -0.010930881450457744, 'd_0_1_18': 0.034010514165168 48, 'd_0_1_19': 0.004614809528294239, 'd_0_1_20': -0.04885125327163622
5, 'd_0_1_21': 0.0062046099380096, 'd_0_1_22': 0.04884142969950865, 'd_0_1_23': -0.008749893700263134, 'd_0_1_24': -0.03794721714309064, 'd_0_1_25': -0.03495434668603256, 'd_0_1_26': -0.024036232
694417584, 'd_0_1_27': -0.0015812201599157832, 'd_0_2_0': -0.005618714491758811, 'd_0_2_1': 0.015676336035307242, 'd_0_2_2': -0.02319188579670174, 'd_0_2_3': 0.013486602274012188, 'd_0_2_4': -0.0
14873152724967 01, 'd_0_2_5': -0.049323785012075176, 'd_0_2_6': 0.003399627762036915, 'd_0_2_7': -0.022566999401791 01, 'd_0_2_8': 0.016475657339745403, 'd_0_2_9': -0.028027877831922222, 'd_0_2_1
0': 0.024350010370076322, 'd_0_2_11': -0.016254747621216162, 'd_0_2_12': 0.04875845960410073, 'd_0_2_13': 0.04887796032795677, 'd_0_2_14': -0.037185391223993086, 'd_0_2_15': -0.02109145865491624,
'd_0_2_16': 0.04949908852780874, 'd_0_2_17': 0.002709690153261314, 'd_0_2_18': 0.04106131791512811, 'd_0_2_19': 0.011878311773878235, 'd_0_2_20': -0.004879442297517987, 'd_0_2_21': 0.010973296014
053668, 'd_0_2_22': -0.04898690009460824, 'd_0_2_23': 0.031951996666598315, 'd_0_2_24': 0.04468638415190866, 'd_0_2_25': -0.03919985217357931, 'd_0_2_26': -0.03912652389236371, 'd_0_2_27': 0.0452
900590000234126, 'd_0_3_0': -0.024174955340581495, 'd_0_3_1': 0.04015661480257776, 'd_0_3_2': 0.048119050744989844, 'd_0_3_3': 0.020888071755946494, 'd_0_3_4': 0.006058920065387177, 'd_0_3_5': 0.0
45701674144825566, 'd_0_3_6': 0.04244053505418814, 'd_0_3_7': 0.011719966449881526, 'd_0_3_8': 0.028218177009384407, 'd_0_3_9': 0.019507852342070267, 'd_0_3_10': 0.039682022260532315, 'd_0_3_11':
-0.007443262416216581, 'd_0_3_12': -0.0030752338215108907, 'd_0_3_13': 0.035306376649659 51, 'd_0_3_14': -0.0022822042054352135, 'd_0_3_15': -0.028155101551951 99, 'd_0_3_16': -0.001518413965149255
4, 'd_0_3_17': -0.013314959853501021, 'd_0_3_18': 0.003504432230112667, 'd_0_3_19': 0.029914949711499055, 'd_0_3_20': 0.0023834148898164015, 'd_0_3_21': -0.012183921912513343, 'd_0_3_22': -0.03976
204351868539, 'd_0_3_23': -0.047067037707233, 'd_0_3_24': 0.025529937170663805, 'd_0_3_25': -0.0016573785481718492, 'd_0_3_26': 0.04486027565262524, 'd_0_3_27': -0.006495494952256063, 'd_0_4_0':
-0.037912998398588485, 'd_0_4_1': 0.0072465369984772016, 'd_0_4_2': 0.04802508231114855, 'd_0_4_3': -0.02078182162524841, 'd_0_4_4': -0.012638428278296913, 'd_0_4_5': -0.042631492453819225, 'd_0_
4_6': 0.011781148747205782, 'd_0_4_7': -0.032055067407741 11, 'd_0_4_8': 0.024054392349926262, 'd_0_4_9': -0.040441818360260035, 'd_0_4_10': -0.0034200199318815266, 'd_0_4_11': -0.015106270390431 2
9, 'd_0_4_12': 0.0072896458661871125, 'd_0_4_13': -0.025492887423311118, 'd_0_4_14': -0.004273117795534721, 'd_0_4_15': -0.004257113614486642, 'd_0_4_16': -0.034156173338392076, 'd_0_4_17': 0.041
143826576345924, 'd_0_4_18': 0.0342308461233901, 'd_0_4_19': 0.045667853606460695, 'd_0_4_20': 0.049079789442504185, 'd_0_4_21': 0.04585072429324559, 'd_0_4_22': -0.006645997505207203, 'd_0_4_2
3': 0.023647495147447 6, 'd_0_4_24': -0.007502685867226987, 'd_0_4_25': 0.026712438928862672, 'd_0_4_26': -0.020778511989404516, 'd_0_4_27': -0.010084638045584036, 'd_0_5_0': 0.02762304733097752,
'd_0_5_1': -0.027250501992987966, 'd_0_5_2': 0.0212843519504681, 'd_0_5_3': -0.036759686707342566, 'd_0_5_4': 0.03773123322986925, 'd_0_5_5': 0.002773521709295901, 'd_0_5_6': 0.03064672100890346
3, 'd_0_5_7': 0.03878647475034839 6, 'd_0_5_8': 0.02785037597441274, 'd_0_5_9': -0.03920197974362875, 'd_0_5_10': -0.048075553230873175, 'd_0_5_11': -0.0367111883339759, 'd_0_5_12': -0.01572672576
797629, 'd_0_5_13': -0.014269287720923805, 'd_0_5_14': -0.027320330918621096, 'd_0_5_15': -0.034608466143597284, 'd_0_5_16': 0.016236863866469914, 'd_0_5_17': 0.03630927565897889, 'd_0_5_18': -0.
033893132855334 24, 'd_0_5_19': 0.0057303939952786 03, 'd_0_5_20': -0.004491236684922774, 'd_0_5_21': 0.003402259886465163, 'd_0_5_22': 0.033561763265906 53, 'd_0_5_23': 0.013736960045182 92, 'd_0_5_
24': 0.021348561703713534, 'd_0_5_25': 0.011527964523556464, 'd_0_5_26': -0.018132675971463057, 'd_0_5_27': 0.022484647919899875, 'd_0_6_0': -0.024544046204643424, 'd_0_6_1': 0.002495273240634643
4, 'd_0_6_2': 0.04174051083644047, 'd_0_6_3': 0.026755237879158608, 'd_0_6_4': 0.008055281986821441, 'd_0_6_5': -0.04738757922197593, 'd_0_6_6': -0.049375110031347674, 'd_0_6_7': -0.034358688725
0502, 'd_0_6_8': 0.047015004731091 88, 'd_0_6_9': 0.03909251874877322, 'd_0_6_10': -0.045411121094170 74, 'd_0_6_11': 0.033997813294832 74, 'd_0_6_12': 0.049447707570451596, 'd_0_6_13': -0.017785472
570536254, 'd_0_6_14': -0.043489060256859 39, 'd_0_6_15': 0.023095739522913716, 'd_0_6_16': -0.034475924714703 01, 'd_0_6_17': 0.030923532029025994, 'd_0_6_18': -0.036840022081690176, 'd_0_6_19': -
0.038366730514606 41, 'd_0_6_20': 0.009114908992711 82, 'd_0_6_21': 0.008244722823564796, 'd_0_6_22': 0.015467863231927684, 'd_0_6_23': 0.035677677421161 39, 'd_0_6_24': 0.023047379658030526, 'd_0_6_
```

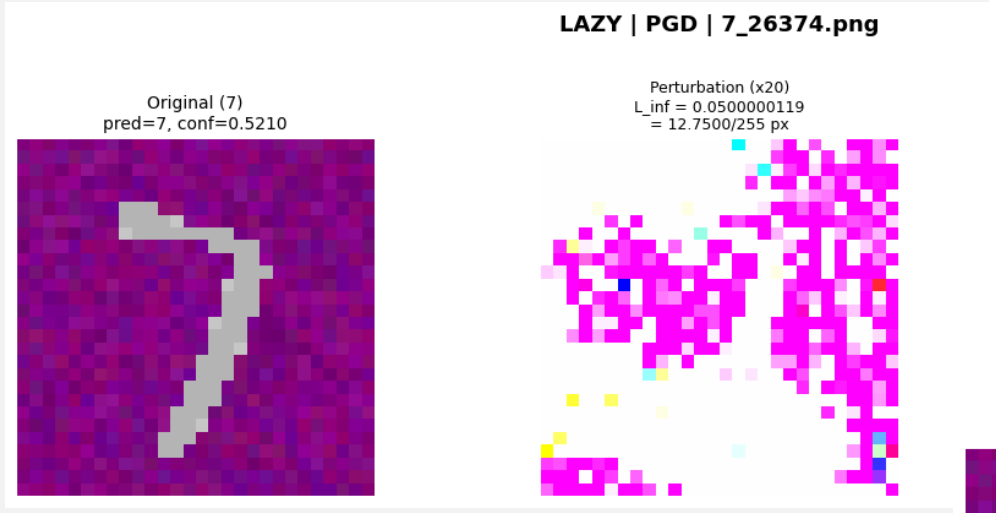
FINALLY ATTACKED! TARGETEDLLLY!!!!: with pgd ,tgdsm,deepfool

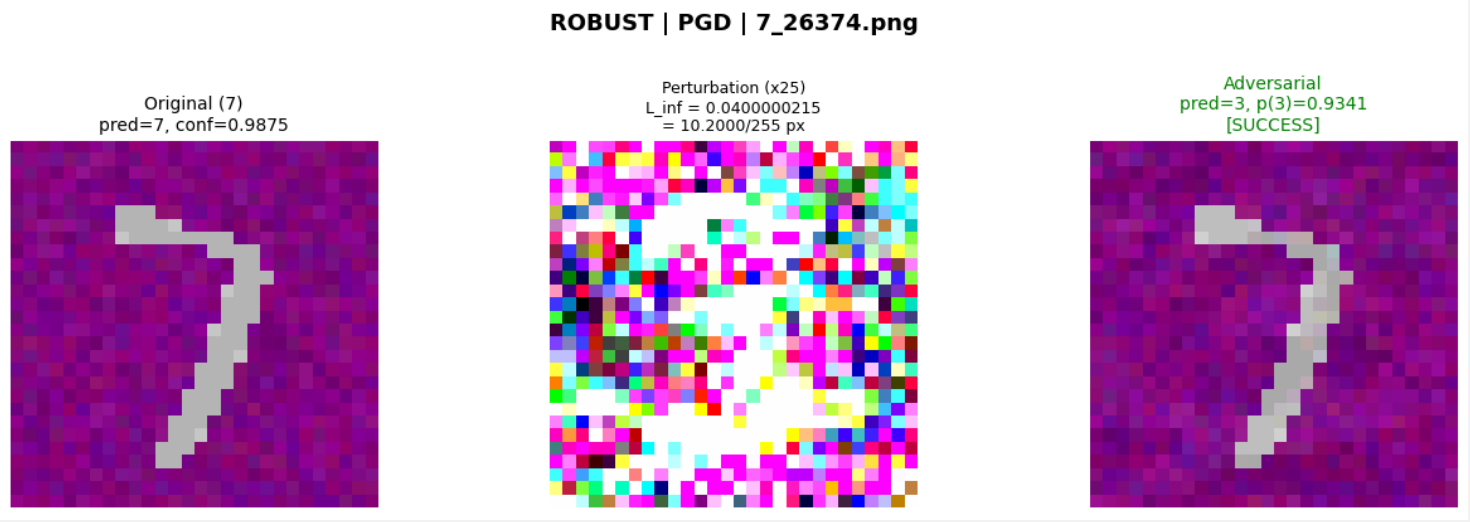
up on selecting top5 images of both models predict 7 with 2nd-choice=3: before THAT I GAVE UP AND THOUGHT FINALLY ,ITS NOT OVER UNTIL I WIN !

```
Total 7-images: 6265
Both predict 7 with 2nd-choice=3: 352

Top-5 selected images:
   #   p(3)_lazy   p(3)_robust   conf_lazy   conf_robust  file
   1    0.019028       0.010362      0.9492        0.9845  7_31850.png
                       0.009425      0.5210        0.9875  7_26374.png
                       0.006559      0.9557        0.9913  7_23946.png
                       0.010650      0.9954        0.9880  7_06050.png
                       0.037047      0.9962        0.9511  7_18322.png
```

YESSSSS

LETS GO::: L_inf= 0.040000021



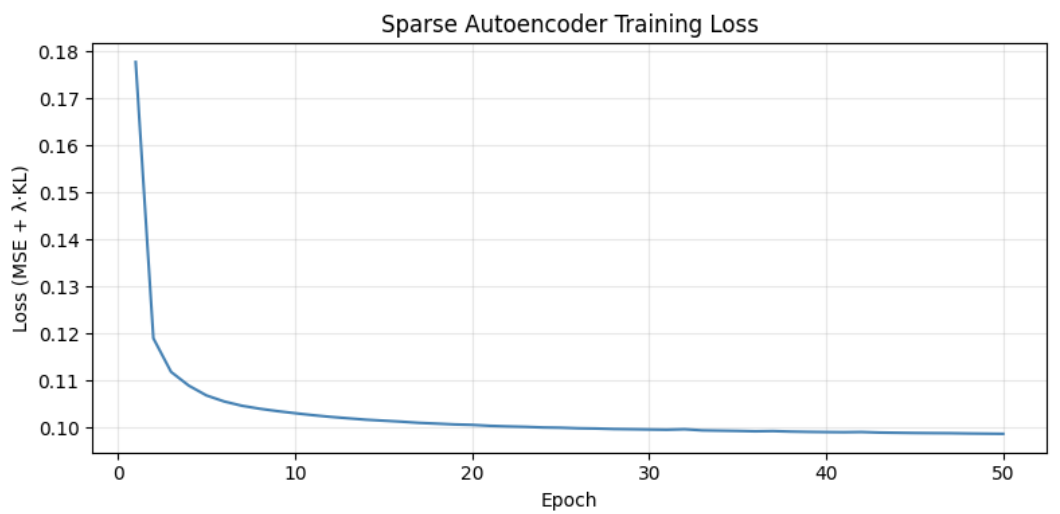


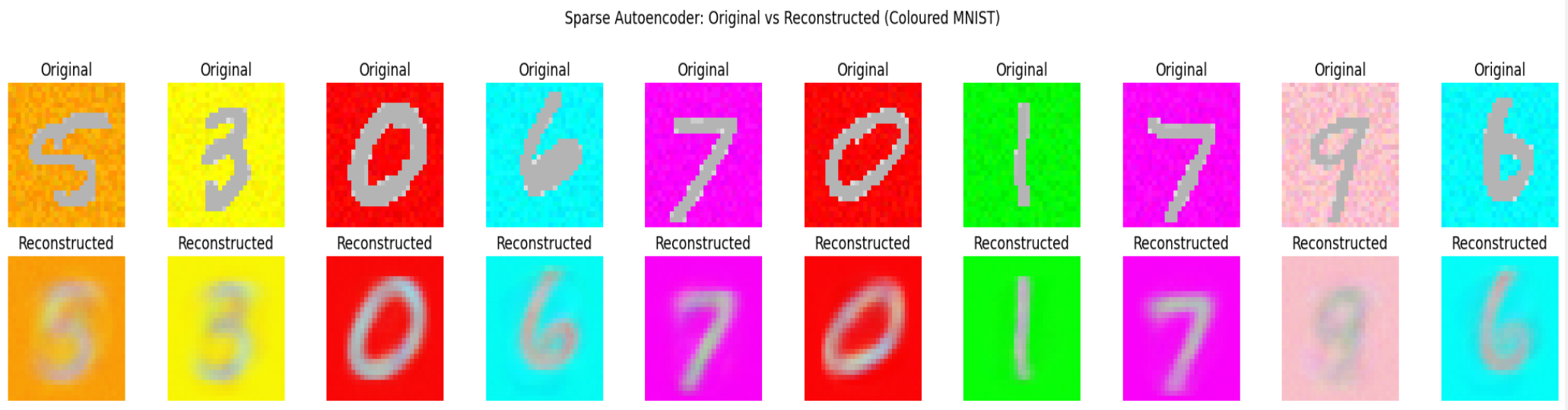
```
================================================================================
  SUMMARY: Targeted Attack 7 -> 3  |  eps < 0.05  |  p(3) > 90%
================================================================================
   #  Image              LAZY best   LAZY p(3)   LAZY ok   ROBUST best    ROB p(3)    ROB ok
  --------------------------------------------------------------------------------
   1  7_31850.png              PGD      0.1998      FAIL           PGD      0.9461      PASS
   2  7_26374.png              PGD      0.4985      FAIL           PGD      0.9341      PASS
   3  7_23946.png              PGD      0.2904      FAIL           PGD      0.9252      PASS
   4  7_06050.png              PGD      0.0817      FAIL           PGD      0.9465      PASS
   5  7_18322.png              PGD      0.0620      FAIL           PGD      0.9457      PASS
  --------------------------------------------------------------------------------
  TOTAL SUCCESSFUL                        0/5                                5/5

  The ROBUST model was EASIER to fool (5/5 vs 0/5).
  The lazy model relies on color shortcuts — pixel noise targets shape, which it ignores.
```

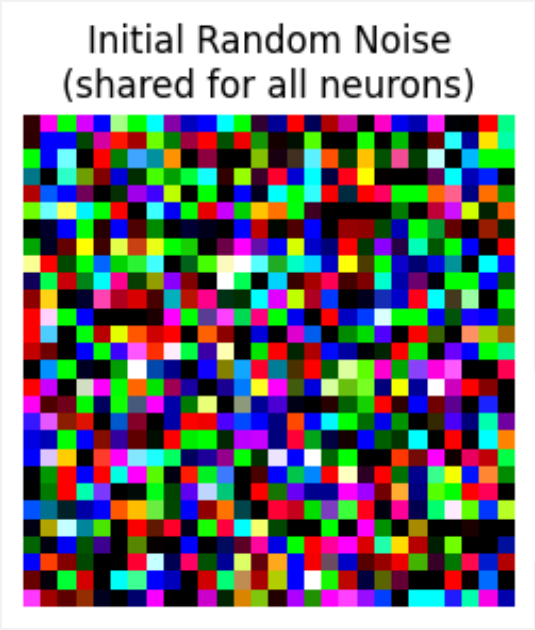
YES LAZY IS CRAZY IN LEARNING COLOURS BUT ROBUST IS ALSO CRAZZY IN GETTING ATTACKED

Task 6: The Decomposition: didn't use latent subspace index but yeah made it bit :

SAE: 2352 → 128 → 2352: WITH @ KLDIVERGENCE

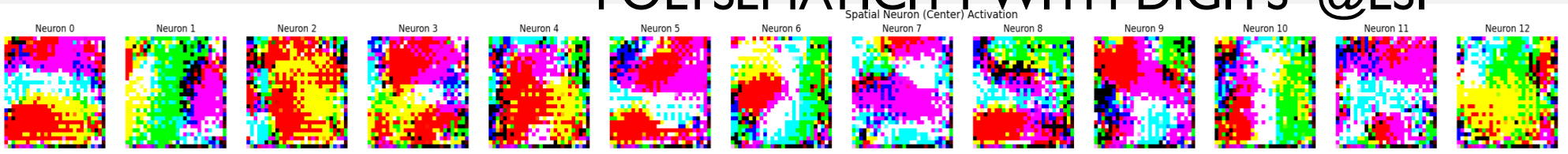


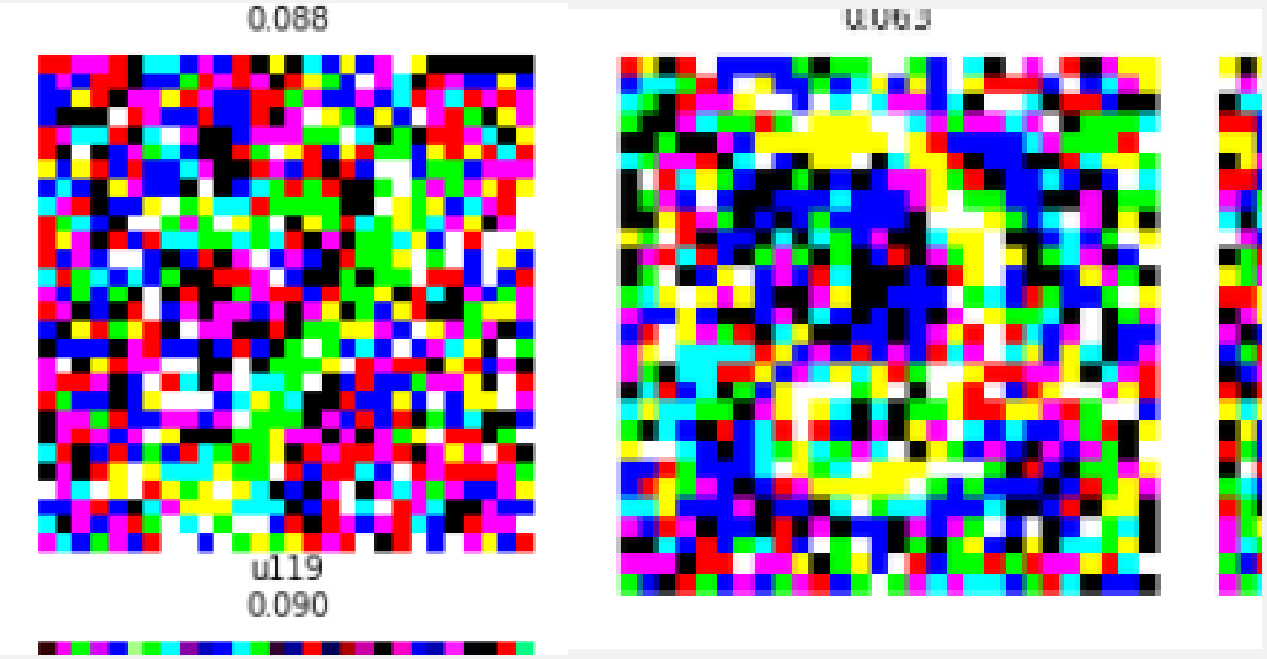


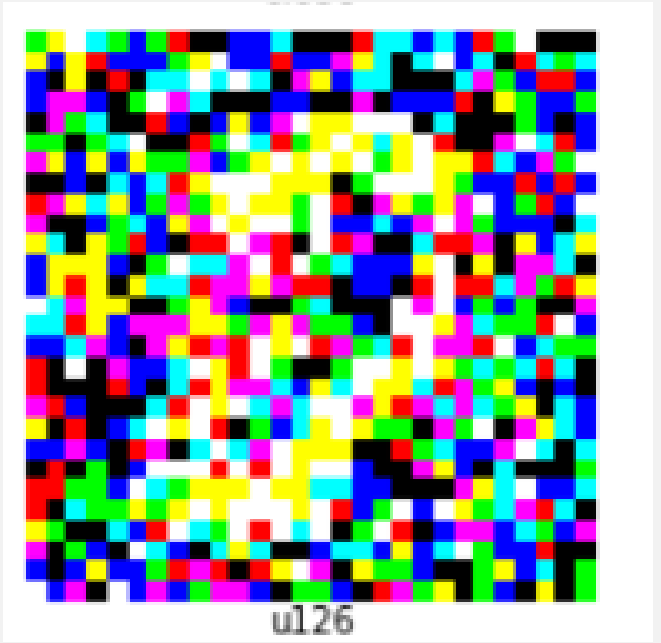


NOW HOOKED WRSPT TO BOTTLE NECK NEURONS

YES SOMETHING TAKES TIME: FROM COLORS, TEXTURES ..TO
POLYSEMATICITY WITH DIGITS @LSI







THANK YOU SO MUCH TEAM PRECOG!! Task was INSANE!!!!!

my eyes are waiting to work under you team PRECOG #pk sir #saralai …IT's NOT OVER until I win!…

references: Selvaraj gradcam,ANDREJ KARPARTHY convetjs,nptel lectures….