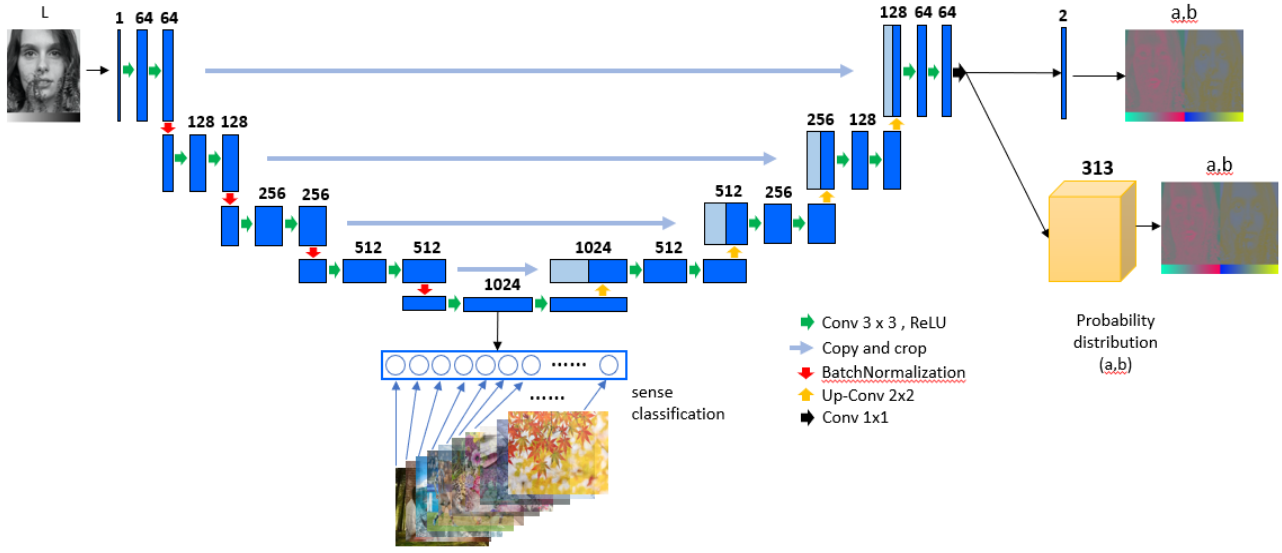
# REPORT PROBLEMS IN OUR MODEL

**Link GitHub**: https://github.com/tramtran2/prlab_image_colorization

## Our Problems:

- **Accuracy is not convergence**
- **Red noise in colorized image**

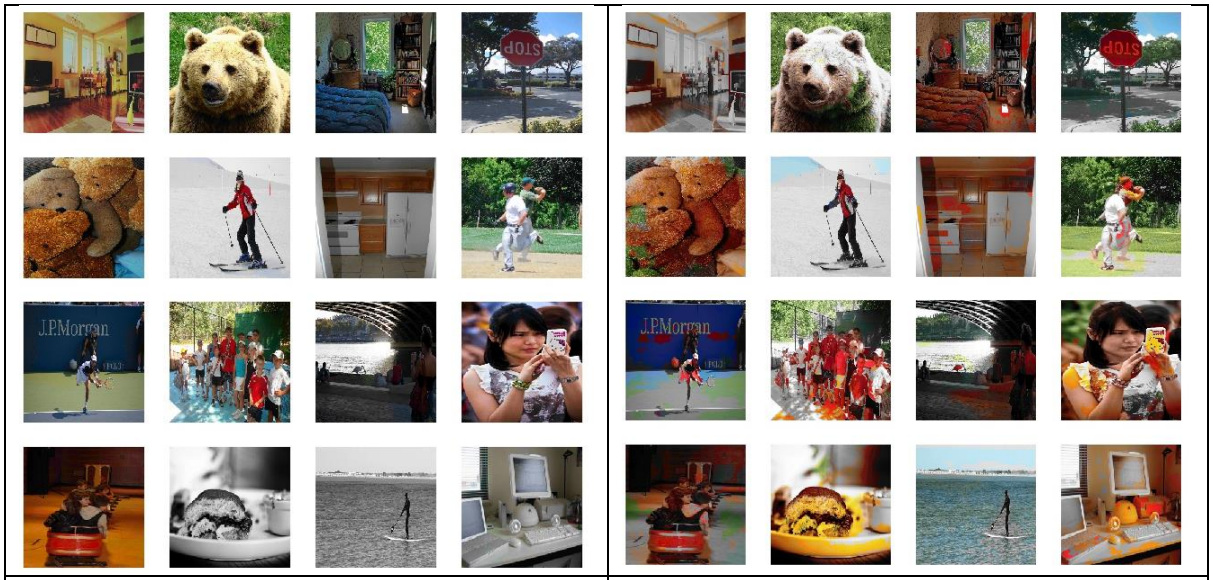
## Our Models



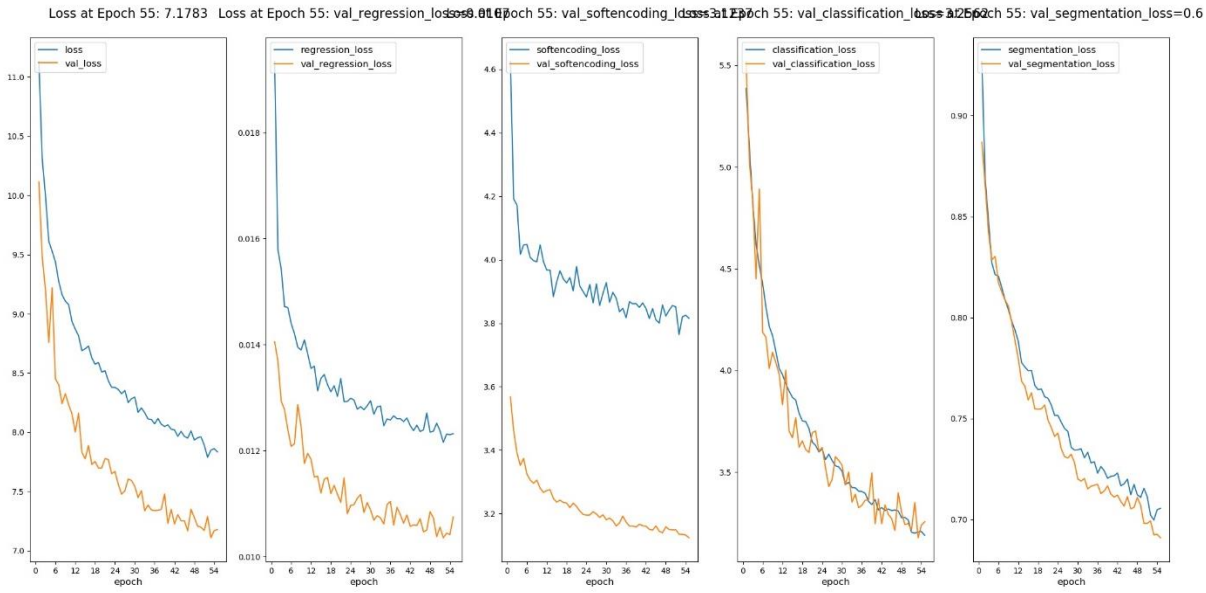

- Input: grayscale image
- Output:
    - Colorization image using regression
    - Colorization image using category cross-entropy with soft encoding with 313 quantization bins
    - Classification branch with scene classification for regularization

## Training History

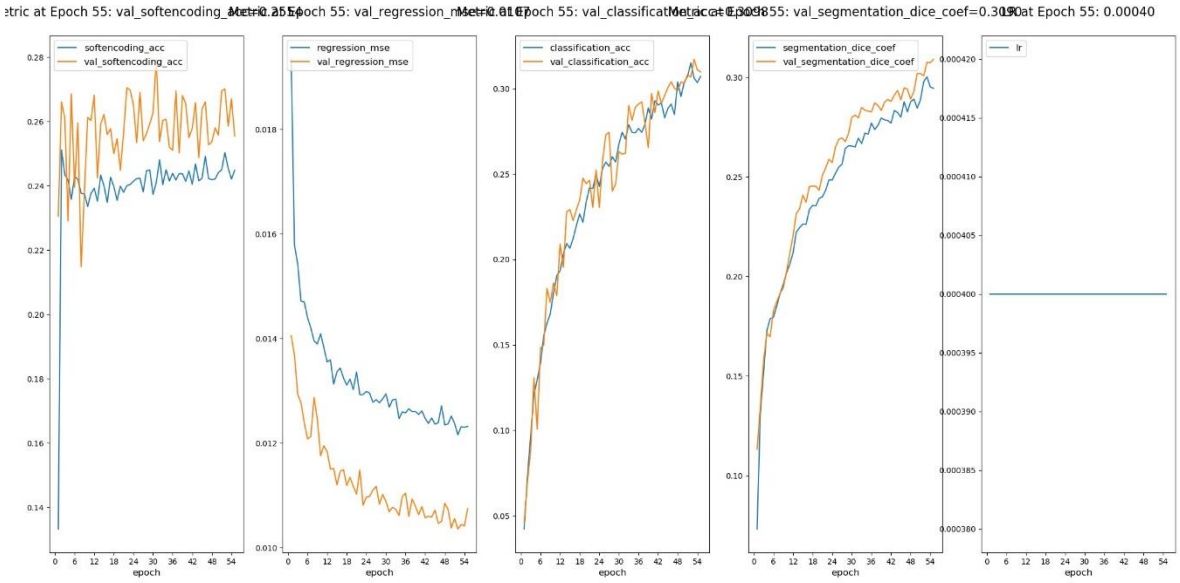
- Using RMSProp, Adam with Step Decay, or Constant or Cycle Learning Rate
- Training and validating on Coco-Stuff



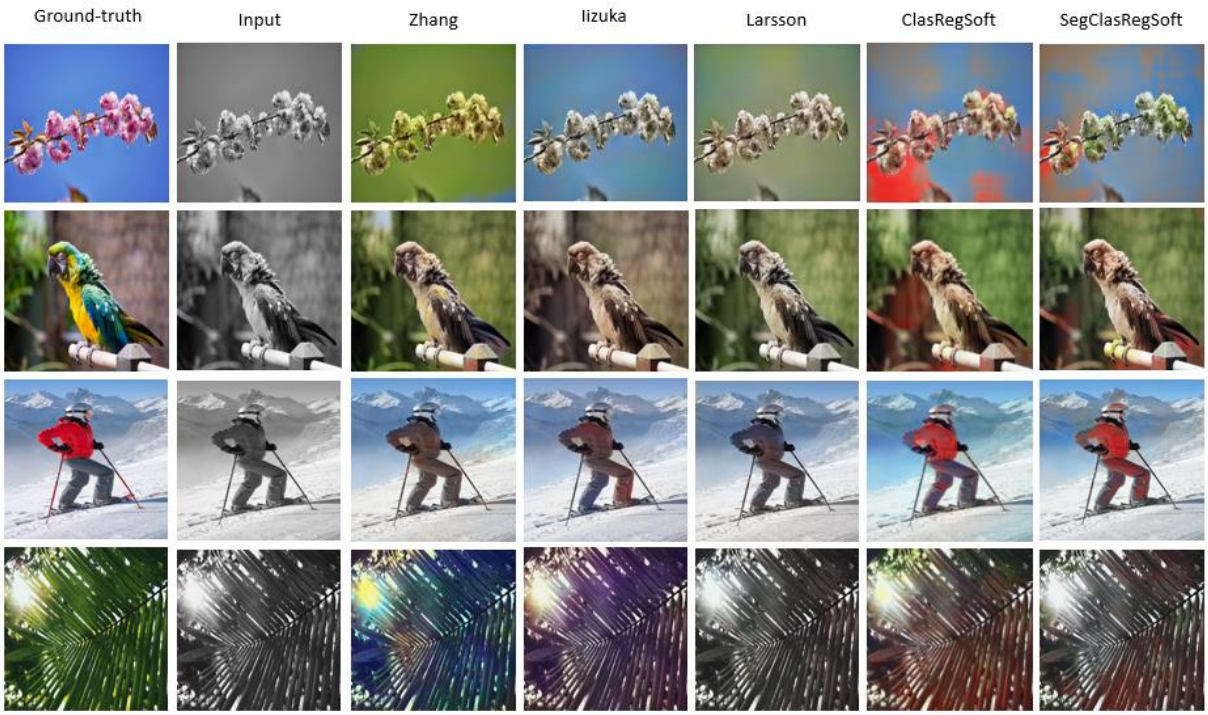
| Ground-truth | ***Results (with Red Noise and some regions are not colorized)*** |
|---|---|




- ***Loss of soft-encoding is good for convergence (image above – column 3) but the accuracy is noise with around 0.32.***

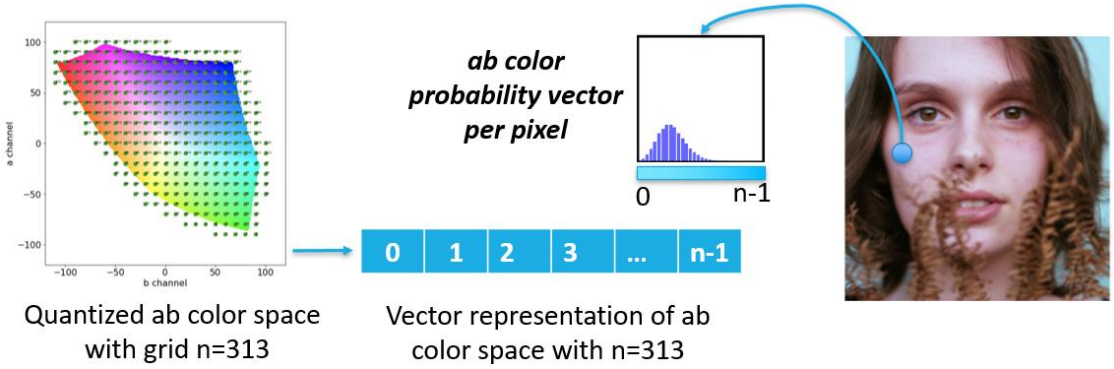



**Testing with collecting images:**




***Red Noise in our method (last two columns)***

## Soft-Encoding




**Implementations:**

For **pts_in_hull**: getting from Richard Zhang works (https://richzhang.github.io/colorization/)

**Soft-Encoding:**

```
def load_nn_finder(pts_in_hull_path, nb_neighbors = 5):
    # Load the array of quantized ab value
    q_ab = np.load(pts_in_hull_path)
    nn_finder = nn.NearestNeighbors(n_neighbors=nb_neighbors, algorithm='ball_
tree').fit(q_ab)
    return q_ab, nn_finder
# load_nn_finder

def get_soft_encoding(image_Lab, nn_finder, nb_q, sigma_neighbor = 5):
    """
    image_Lab = read_image("...")["res_image_Lab"]
    q_ab, nn_finder = load_nn_finder("pts_in_hull.npy", nb_neighbors = 5)
    y = get_soft_encoding(image_Lab, nn_finder, nb_q = q_ab.shape[0], sigma_ne
ighbor = 5)
    """
    # get and normalize image_ab
    # due to preprocessing weighted with minus 128
    image_ab = image_Lab[:, :, 1:].astype(np.int32) - 128

    h, w = image_ab.shape[:2]
    a = np.ravel(image_ab[:, :, 0])
    b = np.ravel(image_ab[:, :, 1])
    ab = np.vstack((a, b)).T

    # Get the distance to and the idx of the nearest neighbors
    dist_neighb, idx_neigh = nn_finder.kneighbors(ab)

    # Smooth the weights with a gaussian kernel
    wts = np.exp(-dist_neighb ** 2 / (2 * sigma_neighbor ** 2))
    wts = wts / np.sum(wts, axis=1)[:, np.newaxis]
```

```
    # format the target
    y = np.zeros((ab.shape[0], nb_q))
    idx_pts = np.arange(ab.shape[0])[:, np.newaxis]
    y[idx_pts, idx_neigh] = wts

    y = y.reshape(h, w, nb_q)

    return y
# get_soft_encoding
```

**Decode with annealing:**

```
def decode_soft_encoding_image_v1(x_image, y_image, q_ab, epsilon = 1e-
8, T = 0.38, x_post_fn = None, usegpu = False, **kwargs):
    """
    q_ab = np.load(os.path.join(module_dir, "data", "pts_in_hull.npy").replace("\\", "
/"))
    x_image: L channel (batchsize, height, width, 1) --> gray
    y_image: softencoding of ab channel (height, width, nb_q)
    """
    if len(x_image.shape) == 2:
        x_batch_image = x_image.reshape((1,) + x_image.shape[0:2] + (1,))
    elif len(x_image.shape) == 3 and x_image.shape[2]==1:
        x_batch_image = x_image.reshape((1,) + x_image.shape[0:3])
    else:
        assert("Invalid Gray Image with shape (w, h) or (w, h, 1)")
    # if
    y_batch_image = y_image.reshape((1,) + y_image.shape[0:3])
    y_batch_RGB = decode_soft_encoding_batch_image_v1(x_batch_image, y_batch_image, q_
ab, epsilon, T, x_post_fn, usegpu = usegpu)
    return y_batch_RGB.reshape(y_batch_RGB.shape[1:])
# decode_image_v1
```

**Decode without annealing:**

```
def decode_soft_encoding_image_v0(x_image, y_image, q_ab, x_post_fn = None, **
kwargs):
    """
    q_ab = np.load(os.path.join(module_dir, "data", "pts_in_hull.npy").replace
("\\", "/"))
    x_image: L channel (batchsize, height, width, 1) --> gray
    y_image: softencoding of ab channel (height, width, nb_q)
    """
    if len(x_image.shape) == 2:
        x_batch_image = x_image.reshape((1,) + x_image.shape[0:2] + (1,))
    elif len(x_image.shape) == 3 and x_image.shape[2]==1:
        x_batch_image = x_image.reshape((1,) + x_image.shape[0:3])
    else:
```

```
        assert("Invalid Gray Image with shape (w, h) or (w, h, 1)")
    # if
    y_batch_image = y_image.reshape((1,) + y_image.shape[0:3])
    y_batch_RGB = decode_soft_encoding_batch_image_v0(x_batch_image, y_batch_i
mage, q_ab, x_post_fn)
    return y_batch_RGB.reshape(y_batch_RGB.shape[1:])
# decode_image_v0
```

## Loss

- **Scene-context classification:** Category Cross-Entropy (CCE) loss:

$$CCE(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log \hat{y}_i$$

Where C is the number of scene, $y_i$/ $\hat{y}_i$ is the ground-truth/predicted scene probability.

- **Pixel Classification of ab color distribution**: Weighted Category Cross-Entropy Loss:

$$CCE(y, \hat{y}) = -\sum_{h,w} v(y_{h,w}) \sum_{i=0}^{N-1} y_{h,w,i} \log \hat{y}_{h,w,i}$$

Where h, w is the height and width of image, N is the number of quantized colors of ab color distribution, $v(\boldsymbol{y_{h,w}})$ **is the weighted of color-class at pixel (h,w) to encourage the rare-color**, $y_{h,w,i}$/ $\hat{y}_{h,w,i}$ is the ground-truth/prediction probability of the soft-encoding color i at pixel (h,w).

- **Regression ab channel**: Using Mean Square Error (MSE) Loss:

$$MSE(y, \hat{y}) = \frac{1}{2hw} \sum_{h,w} \left\| y_{h,w,ab} - \hat{y}_{h,w,ab} \right\|_2^2$$

Where $y_{h,w,ab}$/ $\hat{y}_{h,w,ab}$ is the ground-truth/prediction of ab values at pixel (h,w)

**Implementations**:

```
def build_categorical_crossentropy_color_loss(prior_factor_path = "prior_facto
r.npy", nb_q = 313, train_session = None):
    # Load the color prior factor that encourages rare colors
    prior_factor = None
    if prior_factor_path is None:
        prior_factor = np.ones(nb_q).astype(np.float32)
    elif os.path.exists(prior_factor_path) == False:
        assert(f'prior_factor_path: {prior_factor_path} is not exists!')
    else:
        prior_factor = np.load(prior_factor_path).astype(np.float32)
    # if
    q = len(prior_factor) # number of bins

    def categorical_crossentropy_color(y_true, y_pred):
        y_true = K.reshape(y_true, (-1, q))
```

```
        y_pred = K.reshape(y_pred, (-1, q))

        idx_max = K.argmax(y_true, axis=1)
        weights = K.gather(prior_factor, idx_max)
        weights = K.reshape(weights, (-1, 1))

        # multiply y_true by weights
        y_true = y_true * weights

        # cross_ent = K.categorical_crossentropy(y_pred, y_true)
        cross_ent = K.categorical_crossentropy(y_true, y_pred)
        cross_ent = K.mean(cross_ent, axis=-1)

        return cross_ent
    # wrapper

    return categorical_crossentropy_color
# build_categorical_crossentropy_color_loss
```