CPEN 455: Deep Learning

Tutorial 9: Conditional PixelCNN++

Felix Fu

University of British Columbia

Winter, Term 2, 2025

Outline

- Intro to Conda environment
- **Go through important functions**
- **How to integrate labels and picture?**
- Potential numerical issue
- Important Hyperparameters
- **How long it takes to train a conditional PixelCNN++?**
- **Interfaces + Takeaways**
- **Questions?**

Intro to Conda Environment

- Install Pytorch
- pip install –r requirements.txt

INSTALL PYTORCH

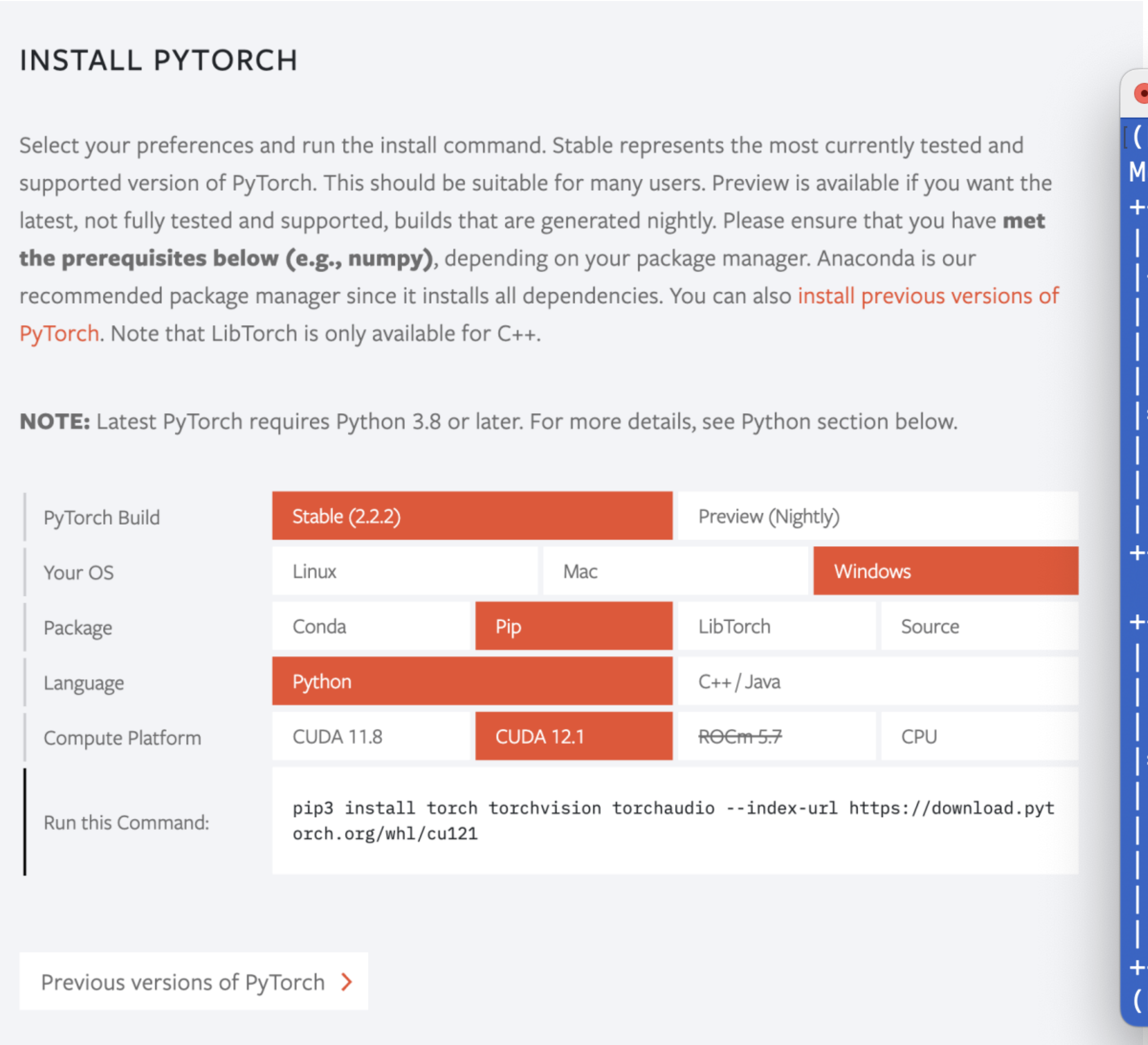
Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

NOTE: Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

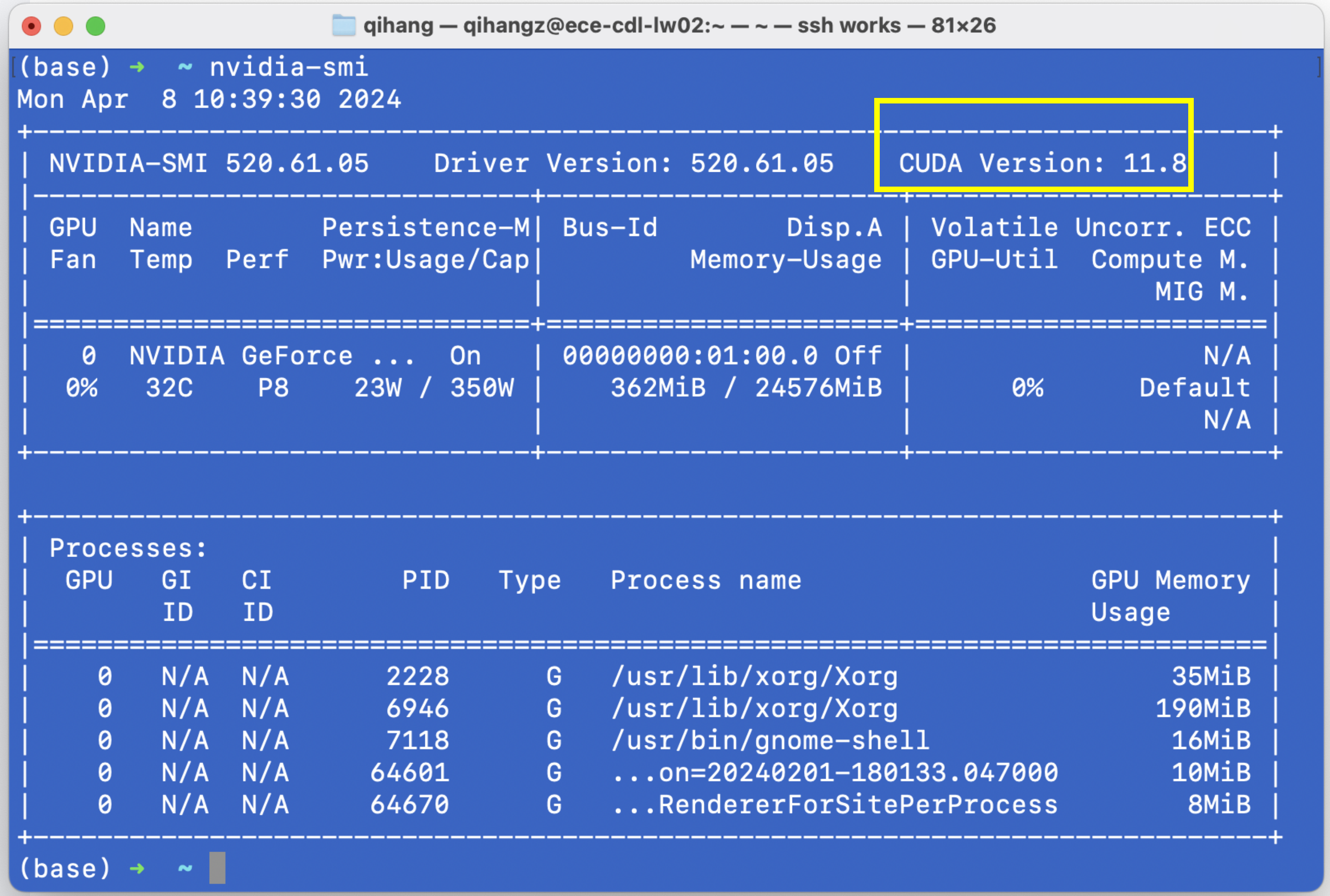
PyTorch Build	Stable (2.2.2)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	~~CUDA 11.8~~	~~CUDA 12.1~~	~~ROCm 5.7~~	Default
Run this Command:	`pip3 install torch torchvision torchaudio`			

Intro to Conda Environment

- Install Pytorch
- pip install –r requirements.txt



nvidia-smi



Dataset.py

dataset.py > ...

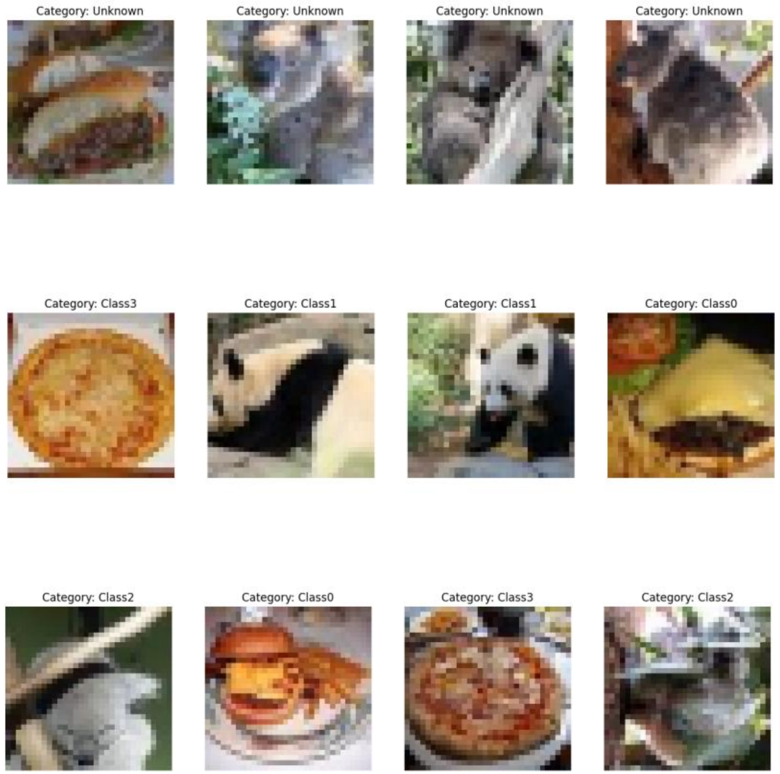
```python
class CPEN455Dataset(Dataset):
    def __init__(self, root_dir, mode='train', transform=None):
        Args:
            root_dir (string): Directory with all the images and labels.
            transform (callable, optional): Optional transform to be applied on a sample.
        """
        ROOT_DIR = './data'
        root_dir = os.path.join(root_dir, mode)
        self.root_dir = root_dir
        self.transform = transform
        self.samples = []  # List to store image paths along with domain and category
        # Walk through the directory structure
        csv_path = os.path.join(ROOT_DIR, mode + '.csv')
        df = pd.read_csv(csv_path, header=None, names=['path', 'label'])
        # Convert DataFrame to a list of tuples
        self.samples = list(df.itertuples(index=False, name=None))
        self.samples = [(os.path.join(ROOT_DIR, path), label) for path, label in self.samples]

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path, category = self.samples[idx]
        if category in my_bidict.values():
            category_name = my_bidict.inverse[category]
        else:
            category_name = "Unknown"
        # print(img_path)
        image = read_image(img_path)  # Reads the image as a tensor
        image = image.type(torch.float32) / 255.  # Normalize to [0, 1]
        if image.shape[0] == 1:
            image = replicate_color_channel(image)
        if self.transform:
          image = self.transform(image)
        return image, category_name
```

Dataset.py

dataset.py M ×

dataset.py > show_images

```python
if __name__ == '__main__':

    transform_32 = Compose([
        Resize((32, 32)),  # Resize images to 32 * 32
        rescaling
    ])
    dataset_list = ['train', 'validation', 'test']

    for mode in dataset_list:
        print(f"Mode: {mode}")
        dataset = CPEN455Dataset(root_dir='./data', transform=transform_32, mode=mode)
        data_loader = DataLoader(dataset, batch_size = 4, shuffle=True)
        # Sample from the DataLoader
        for images, categories in tqdm(data_loader):
            print(images.shape, categories)
            images = torch.round(rescaling_inv(images) * 255).type(torch.uint8)
            show_images(images, categories, mode)
            break  # We only want to see one batch of 4 images in this example
```



pcnn_train.py

dataset.py | pcnn_train.py | utils.py | model.py

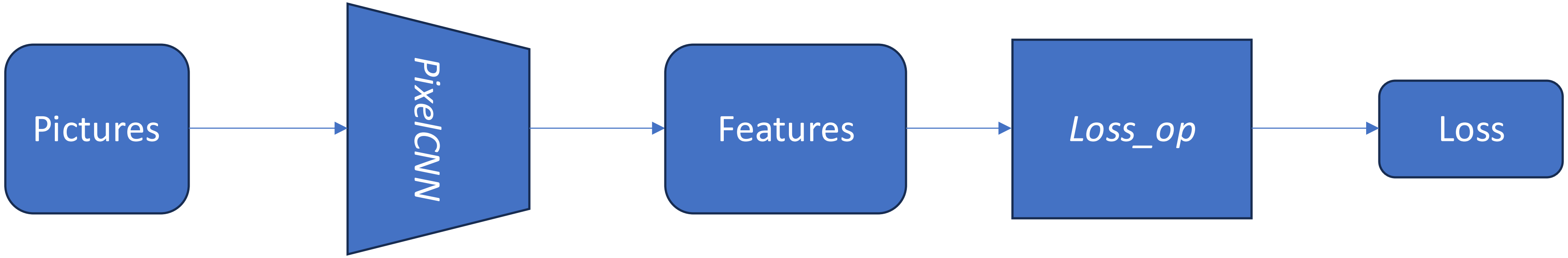
pcnn_train.py > ...

```python

def train_or_test(model, data_loader, optimizer, loss_op, device, args, epoch, mode = 'training'):
    if mode == 'training':
        model.train()
    else:
        model.eval()

    deno =  args.batch_size * np.prod(args.obs) * np.log(2.)
    loss_tracker = mean_tracker()

    for batch_idx, item in enumerate(tqdm(data_loader)):
        model_input, _ = item
        model_input = model_input.to(device)
        model_output = model(model_input)
        loss = loss_op(model_input, model_output)
        loss_tracker.update(loss.item()/deno)
        if mode == 'training':
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    if args.en_wandb:
        wandb.log({mode + "-Average-BPD" : loss_tracker.get_mean()})
        wandb.log({mode + "-epoch": epoch})
```

label

Most important function in whole project:

discretized_mix_logistic_loss(x, l) line36 in utils.py



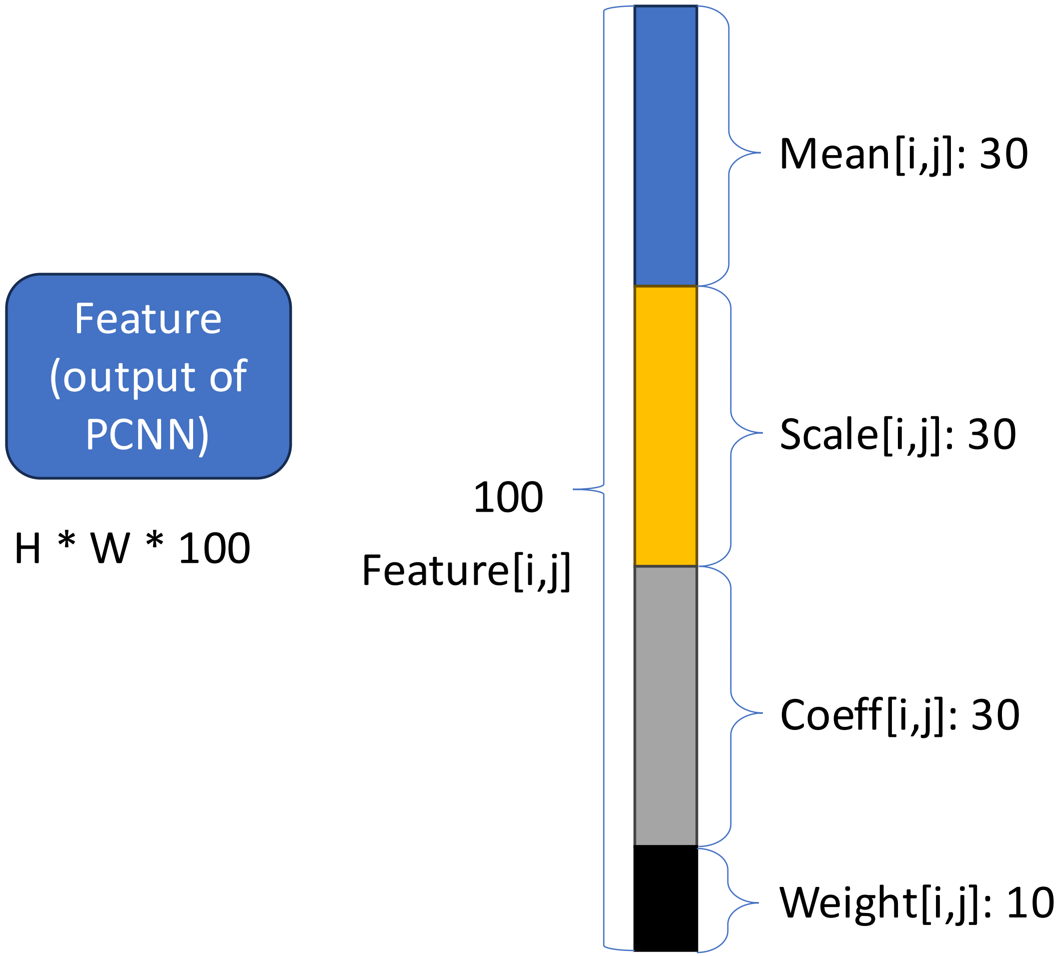
$$D_{KL}(P_{data}, P_\theta) = \sum_z P_{data}(z) \cdot \log \frac{P_{data}(z)}{P_\theta(z)}$$

$$= -\sum_z P_{data}(z) \log P_\theta(z) + \underbrace{\sum_z P_{data}(z) \log P_{data}(z)}_{\text{constant}}$$

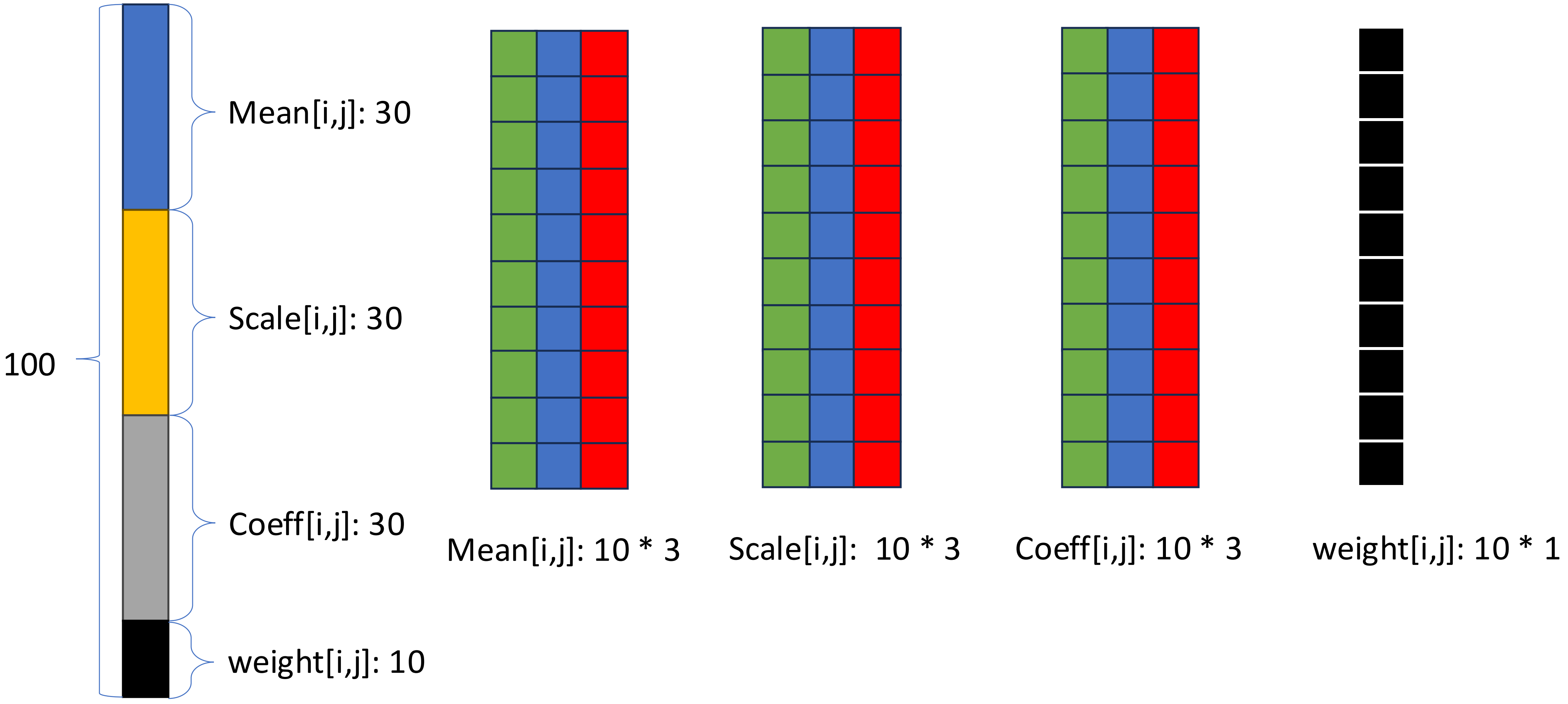
$$\approx -\sum_{i=1}^{N} \log P_\theta(i^{th}\ \text{Picture})$$

Short conclusion: the output of loss function should be of the form NLL

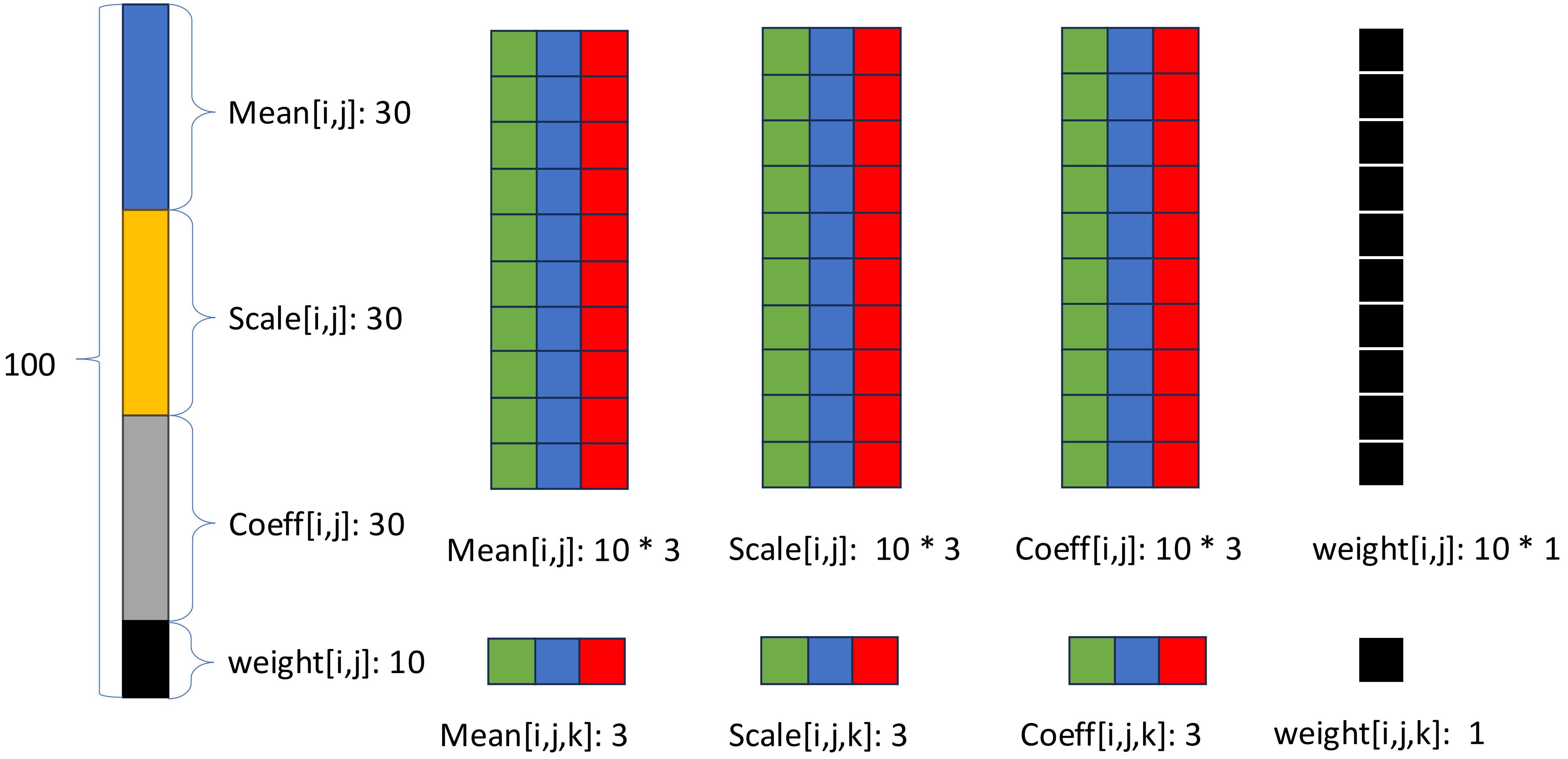
discretized_mix_logistic_loss(x, l)



discretized_mix_logistic_loss(x, l)



discretized_mix_logistic_loss(x, l)



discretized_mix_logistic_loss(x, l)

Mean[i,j,k]: 3 Scale[i,j,k]: 3 Coeff[i,j,k]: 3 weight[i,j,k]: 1

$$\begin{aligned}
p(r_{i,j}, g_{i,j}, b_{i,j} | C_{i,j}) &= P(r_{i,j} | \mu_r(C_{i,j}), s_r(C_{i,j})) \times P(g_{i,j} | \mu_g(C_{i,j}, r_{i,j}), s_g(C_{i,j})) \\
&\quad \times P(b_{i,j} | \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), s_b(C_{i,j})) \\
\mu_g(C_{i,j}, r_{i,j}) &= \mu_g(C_{i,j}) + \alpha(C_{i,j}) r_{i,j} \\
\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) &= \mu_b(C_{i,j}) + \beta(C_{i,j}) r_{i,j} + \gamma(C_{i,j}) g_{i,j}
\end{aligned}$$

discretized_mix_logistic_loss(x, l)

Mean[i,j,k]: 3

Scale[i,j,k]: 3

Coeff[i,j,k]: 3

weight[i,j,k]: 1

Mean[i,j,k,v], v = 0,1,2

Scale[i,j,k,v], v = 0,1,2

Coeff[i,j,k,v], v = 0,1,2

$$
\begin{aligned}
p(r_{i,j}, g_{i,j}, b_{i,j} | C_{i,j}) &= P(r_{i,j} | \mu_r(C_{i,j}), \boxed{s_r(C_{i,j})}) \times P(g_{i,j} | \mu_g(C_{i,j}, r_{i,j}), \boxed{s_g(C_{i,j})}) \\
&\quad \times P(b_{i,j} | \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), \boxed{s_b(C_{i,j})}) \\
\mu_g(C_{i,j}, r_{i,j}) &= \mu_g(C_{i,j}) + \alpha(C_{i,j}) r_{i,j} \\
\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) &= \mu_b(C_{i,j}) + \beta(C_{i,j}) r_{i,j} + \gamma(C_{i,j}) g_{i,j},
\end{aligned}
$$

Note: equation from the original paper has a typo.

discretized_mix_logistic_loss(x, l)

Mean[i,j,k]: 3

Scale[i,j,k]: 3

Coeff[i,j,k]: 3

weight[i,j,k]: 1

Mean[i,j,k,v], v = 0,1,2

Scale[i,j,k,v], v = 0,1,2

Coeff[i,j,k,v], v = 0,1,2

$$
\begin{aligned}
p(r_{i,j}, g_{i,j}, b_{i,j} | C_{i,j}) &= P(r_{i,j} | \boxed{\mu_r(C_{i,j})}, s_r(C_{i,j})) \times P(g_{i,j} | \mu_g(C_{i,j}, r_{i,j}), s_g(C_{i,j})) \\
&\quad \times P(b_{i,j} | \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), s_b(C_{i,j})) \\
\mu_g(C_{i,j}, r_{i,j}) &= \boxed{\mu_g(C_{i,j})} + \boxed{\alpha(C_{i,j}) r_{i,j}} \\
\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) &= \boxed{\mu_b(C_{i,j})} + \boxed{\beta(C_{i,j}) r_{i,j}} + \boxed{\gamma(C_{i,j}) g_{i,j}},
\end{aligned}
$$

discretized_mix_logistic_loss(x, l)

Mean[i,j,k,v], v = 0,1,2 Scale[i,j,k,v], v = 0,1,2 Coeff[i,j,k,v], v = 0,1,2

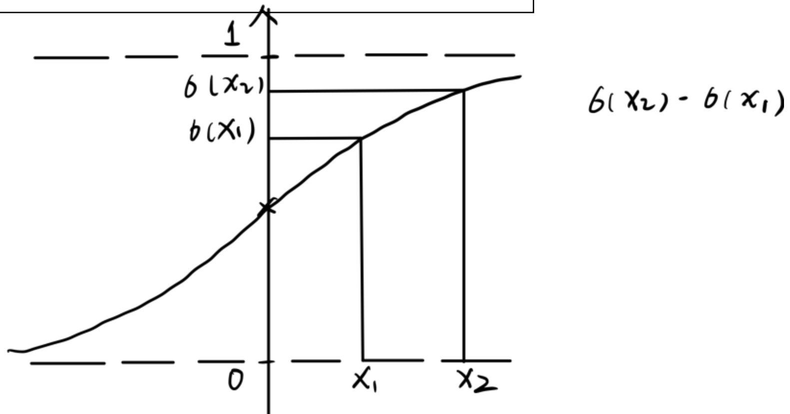
weight[i,j,k]: 1

$$
\begin{aligned}
p(r_{i,j}, g_{i,j}, b_{i,j} | C_{i,j}) &= P(r_{i,j} | \boxed{\mu_r(C_{i,j})}, \boxed{s_r(C_{i,j})}) \times P(g_{i,j} | \mu_g(C_{i,j}, r_{i,j}), \boxed{s_g(C_{i,j})}) \\
&\quad \times P(b_{i,j} | \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), \boxed{s_b(C_{i,j})}) \\
\mu_g(C_{i,j}, r_{i,j}) &= \boxed{\mu_g(C_{i,j})} + \boxed{\alpha(C_{i,j}) r_{i,j}} \\
\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) &= \boxed{\mu_b(C_{i,j})} + \boxed{\beta(C_{i,j}) r_{i,j}} + \boxed{\gamma(C_{i,j}) g_{i,j}},
\end{aligned}
$$

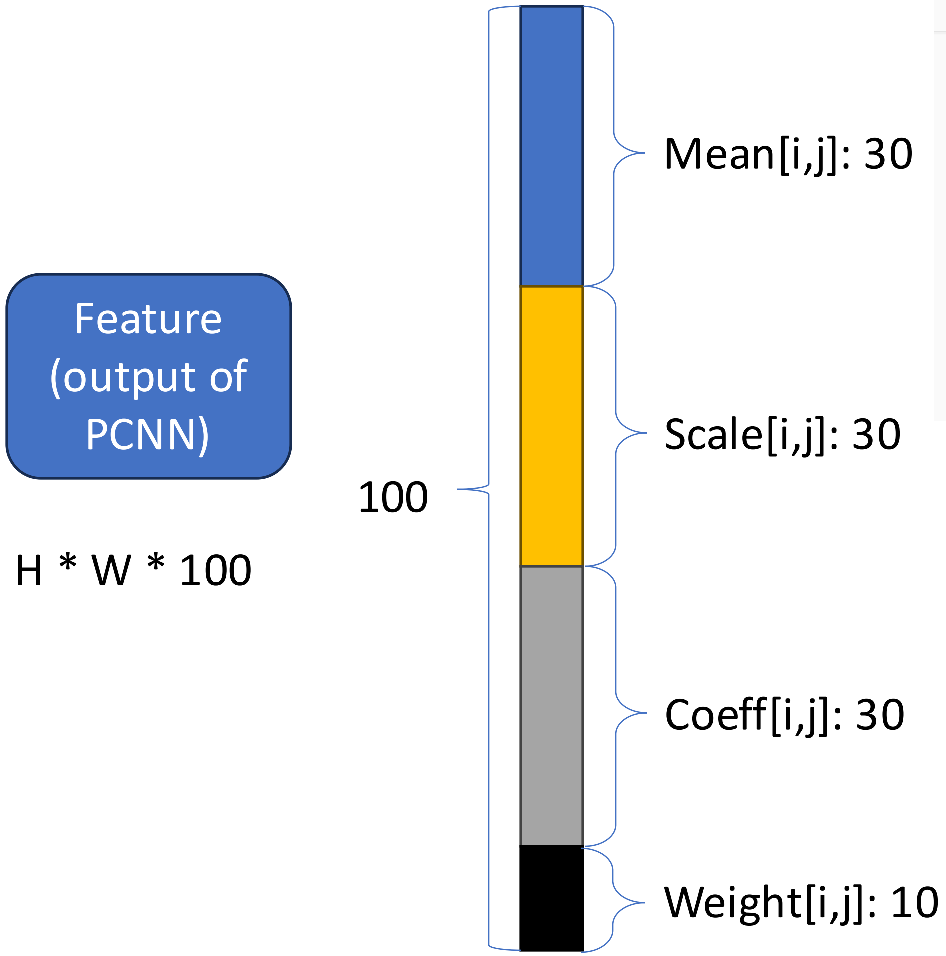
Scale

$$
P(x | \pi, \mu, s) = \sum_{i=1}^{K} \boxed{\pi_i} [\sigma((x + 0.5 - \mu_i)/s_i) - \sigma((x - 0.5 - \mu_i)/s_i)]
$$

Mean



discretized_mix_logistic_loss(x, l)




```python
def discretized_mix_logistic_loss(x, l):

    # here and below: unpacking the params of the mixture of logistics
    nr_mix = int(ls[-1] / 10)
    logit_probs = l[:, :, :, :nr_mix]
    l = l[:, :, :, nr_mix:].contiguous().view(xs + [nr_mix * 3]) # 3 for mean, scale, coef
    means = l[:, :, :, :, :nr_mix]
    # log_scales = torch.max(l[:, :, :, :, nr_mix:2 * nr_mix], -7.)
    log_scales = torch.clamp(l[:, :, :, :, nr_mix:2 * nr_mix], min=-7.)

    coeffs = F.tanh(l[:, :, :, :, 2 * nr_mix:3 * nr_mix])
```

discretized_mix_logistic_loss(x, l)

Mean[i,j,k,v], v = 0,1,2

Coeff[i,j,k,v], v = 0,1,2

$$
\begin{aligned}
p(r_{i,j}, g_{i,j}, b_{i,j} | C_{i,j}) &= P(r_{i,j} | \boxed{\mu_r(C_{i,j})}, s_r(C_{i,j})) \times P(g_{i,j} | \mu_g(C_{i,j}, r_{i,j}), s_g(C_{i,j})) \\
&\quad \times P(b_{i,j} | \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), s_b(C_{i,j})) \\
\mu_g(C_{i,j}, r_{i,j}) &= \boxed{\mu_g(C_{i,j})} + \boxed{\alpha(C_{i,j}) r_{i,j}} \\
\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) &= \boxed{\mu_b(C_{i,j})} + \boxed{\beta(C_{i,j}) r_{i,j}} + \boxed{\gamma(C_{i,j}) g_{i,j};}
\end{aligned}
$$

```python
def discretized_mix_logistic_loss(x, l):

    coeffs = F.tanh(l[:, :, :, :, 2 * nr_mix:3 * nr_mix])
    # here and below: getting the means and adjusting them based on preceding
    # sub-pixels
    x = x.contiguous()      You, 2 weeks ago • first commit
    x = x.unsqueeze(-1) + Variable(torch.zeros(xs + [nr_mix]).to(x.device), requires_grad=False)
    m2 = (means[:, :, :, 1, :] + coeffs[:, :, :, 0, :]
                * x[:, :, :, 0, :]).view(xs[0], xs[1], xs[2], 1, nr_mix)

    m3 = (means[:, :, :, 2, :] + coeffs[:, :, :, 1, :] * x[:, :, :, 0, :] +
                coeffs[:, :, :, 2, :] * x[:, :, :, 1, :]).view(xs[0], xs[1], xs[2], 1, nr_mix)

    means = torch.cat((means[:, :, :, 0, :].unsqueeze(3), m2, m3), dim=3)
```

discretized_mix_logistic_loss(x, l)

```python
def discretized_mix_logistic_loss(x, l):
    centered_x = x - means
    inv_stdv = torch.exp(-log_scales)
    plus_in = inv_stdv * (centered_x + 1. / 255.)
    cdf_plus = F.sigmoid(plus_in)
    min_in = inv_stdv * (centered_x - 1. / 255.)
    cdf_min = F.sigmoid(min_in)
    # log probability for edge case of 0 (before scaling)
    log_cdf_plus = plus_in - F.softplus(plus_in)
    # log probability for edge case of 255 (before scaling)
    log_one_minus_cdf_min = -F.softplus(min_in)
    cdf_delta = cdf_plus - cdf_min  # probability for all other cases
```

$$[\sigma((x + 0.5 - \mu_i)/s_i) - \sigma((x - 0.5 - \mu_i)/s_i)]$$

discretized_mix_logistic_loss(x, l)

```
def discretized_mix_logistic_loss(x, l):
    mid_in = inv_stdv * centered_x
    # log probability in the center of the bin, to be used in extreme c
    # (not actually used in our code)
    log_pdf_mid = mid_in - log_scales - 2. * F.softplus(mid_in)

    # now select the right output: left edge case, right edge case, nor
    # case, extremely low prob case (doesn't actually happen for us)

    # this is what we are really doing, but using the robust version be
    # log_probs = tf.select(x < -0.999, log_cdf_plus, tf.select(x > 0.9

    # robust version, that still works if probabilities are below 1e-5
    # tensorflow backpropagates through tf.select() by multiplying with
    # the 1e-12 in tf.maximum(cdf_delta, 1e-12) is never actually used
    # if the probability on a sub-pixel is below 1e-5, we use an approx
    # based on the assumption that the log-density is constant in the b
    # the observed sub-pixel value

    inner_inner_cond = (cdf_delta > 1e-5).float()
    inner_inner_out  = inner_inner_cond * torch.log(torch.clamp(cdf_de
    inner_cond       = (x > 0.999).float()
    inner_out        = inner_cond * log_one_minus_cdf_min + (1. - inner
    cond             = (x < -0.999).float()
    log_probs        = cond * log_cdf_plus + (1. - cond) * inner_out
```

For numerical stability

discretized_mix_logistic_loss(x, l)

```
def discretized_mix_logistic_loss(x, l):
    log_probs          = torch.sum(log_probs, dim=3) + log_prob_from_logits(logit_probs)

    return -torch.sum(log_sum_exp(log_probs))
```

$$P(x|\pi,\mu,s) = \boxed{\sum_{i=1}^{K}}\boxed{\pi_i}[\sigma((x+0.5-\mu_i)/s_i)-\sigma((x-0.5-\mu_i)/s_i)]$$

$$\begin{aligned} p(r_{i,j},g_{i,j},b_{i,j}|C_{i,j}) &= P(r_{i,j}|\mu_r(C_{i,j}),s_r(C_{i,j}))\times P(g_{i,j}|\mu_g(C_{i,j},r_{i,j}),s_g(C_{i,j})) \\ &\quad \times P(b_{i,j}|\mu_b(C_{i,j},r_{i,j},g_{i,j}),s_b(C_{i,j})) \end{aligned}$$

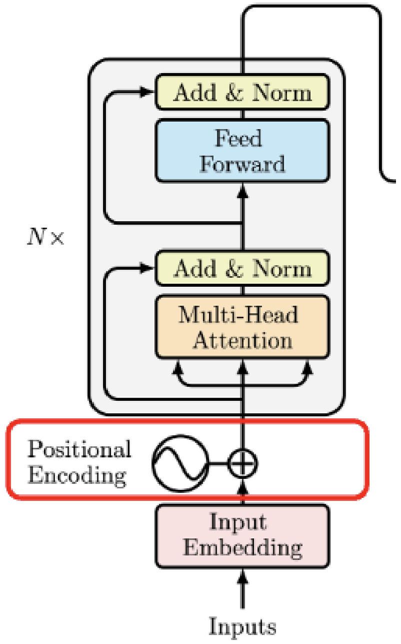
Short conclusion:
If you want to get the log prob of each picture, you should modify the last line

How to integrate labels and picture?

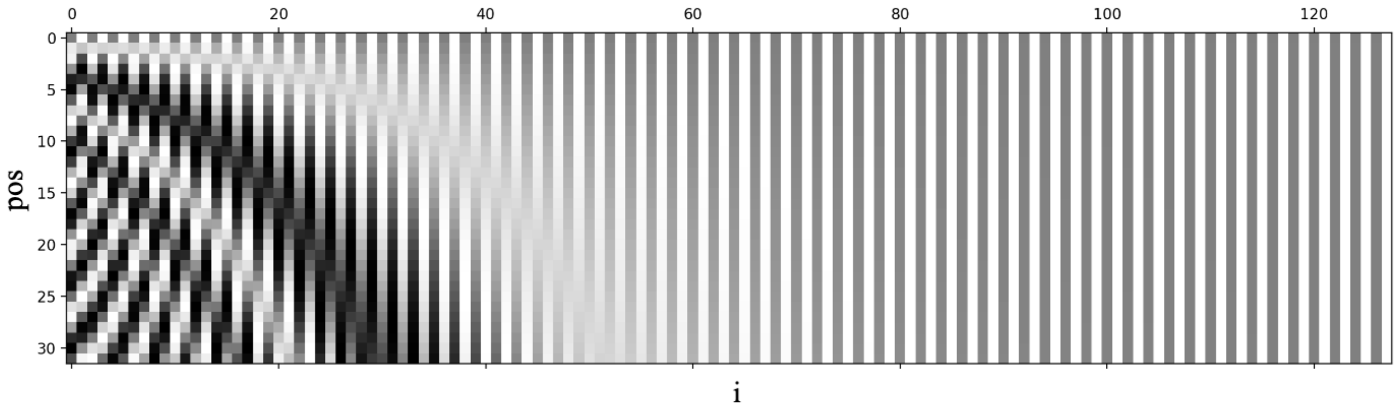
- How to encode labels?
- How to integrate it with pictures?

some hints from Positional Encoding:

1. Encode the label into a tensor (positional encoding in PA2)
2. Integrate it with input embedding



Positional Encoding

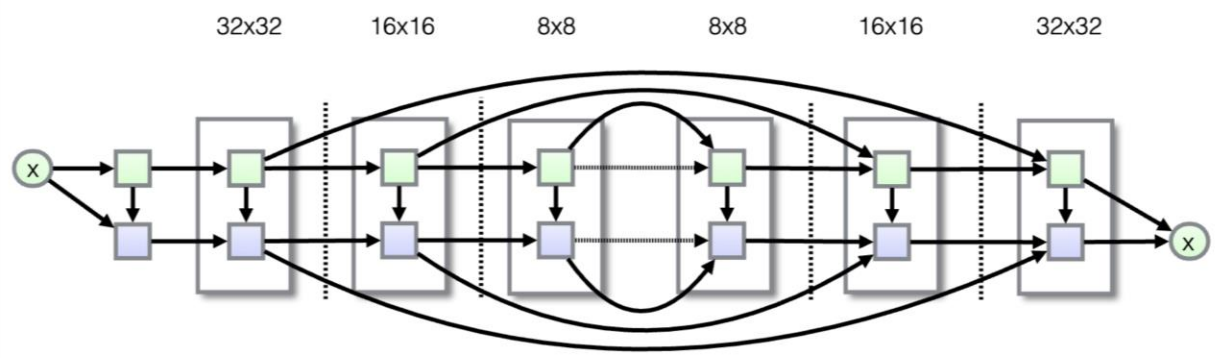


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Text and image integration

Different position has different impact
explore it by yourself.




```python
class PixelCNN(nn.Module):
    def forward(self, x, sample=False):

        ###      UP PASS    ###
        x = x if sample else torch.cat((x, self.init_padding), 1)
        u_list  = [self.u_init(x)]
        ul_list = [self.ul_init[0](x) + self.ul_init[1](x)]
        for i in range(3):
            # resnet block
            u_out, ul_out = self.up_layers[i](u_list[-1], ul_list[-1])
            u_list  += u_out
            ul_list += ul_out

            if i != 2:
                # downscale (only twice)
                u_list  += [self.downsize_u_stream[i](u_list[-1])]
                ul_list += [self.downsize_ul_stream[i](ul_list[-1])]

        ###    DOWN PASS    ###
        u  = u_list.pop()
        ul = ul_list.pop()

        for i in range(3):
            # resnet block
            u, ul = self.down_layers[i](u, ul, u_list, ul_list)

            # upscale (only twice)
            if i != 2 :
                u  = self.upsize_u_stream[i](u)
                ul = self.upsize_ul_stream[i](ul)

        x_out = self.nin_out(F.elu(ul))
```

Potential numerical issue

$$p(i|x) = \frac{p(x|i)}{\sum_{k=1}^{4} p(x|k)}$$

$$p(x|i) = \exp(\log p(x|i))$$

$$\log p(x|i)$$

$$= \sum \log p(x_j | x_{<i}, i)$$

if $\log p(x_j | x_{<j}, i) \sim -8$

$$\log p(x|i) \sim -8 \times 32 \times 32 \times 3 = \sim -24k$$

$\exp(-24k) \longrightarrow$ underflow

$$\log p(i|x) = \log p(x|i) - \underbrace{\log \sum_{k=1}^{4} p(x|k)}_{\text{torch.logsumexp}}$$

$$p(i|x) = \exp(\log p(i|x))$$

Important Hyperparameters

dataset.py | pcnn_train.py | utils.py | model.py

pcnn_train.py > ...

```python

    # model
    parser.add_argument('-q', '--nr_resnet', type=int, default=5,
                        help='Number of residual blocks per stage of the model')
    parser.add_argument('-n', '--nr_filters', type=int, default=160,
                        help='Number of filters to use across the model. Higher = larger model.')
    parser.add_argument('-m', '--nr_logistic_mix', type=int, default=10,
                        help='Number of logistic components in the mixture. Higher = more flexible model')
```

How long it takes to train a conditional PixelCNN++?

https://wandb.ai/qihangz-work/CPEN455HW?nw=nwuserqihangz

Interfaces

- Generation_evaluation.py
  - Fréchet Inception Distance
- Classification_evaluation.py
  - Validation set accuracy
  - Note that test set is for the final grading

Takeaways

- The logistic mixture model is **more stable** and achieves higher log-likelihood scores. It captures pixel dependencies and results in **smoother, more natural image generation.**

- Instead of using a fully autoregressive approach, PCNN++ implements two-stream processing, improving spatial coherence and long-range dependencies.

- Use downsampling to efficiently capture structure at multiple resolutions. Use a **coarse-to-fine hierarchical approach**, modeling a lower-resolution version first, then refining details.

Questions?