AI VIETNAM
All-in-One Course

VAE-based Image Colorization

Nhu-Tai Do
Ph.D. in Computer Science

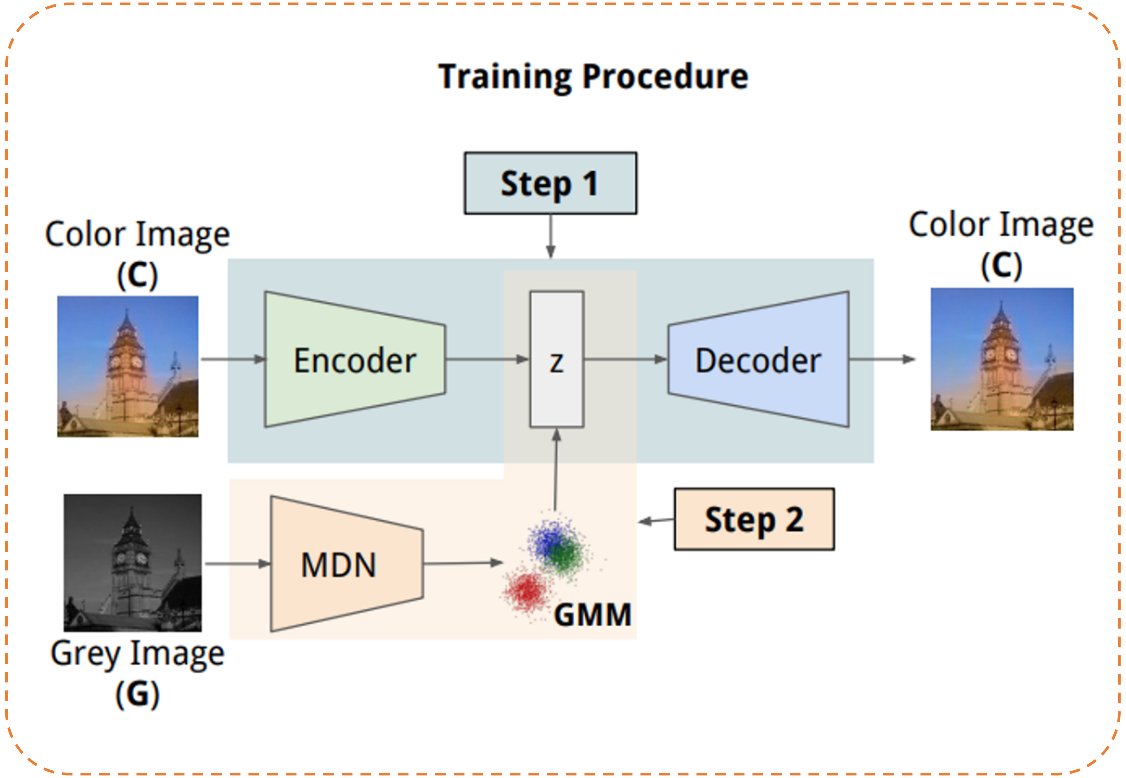
Year 2024

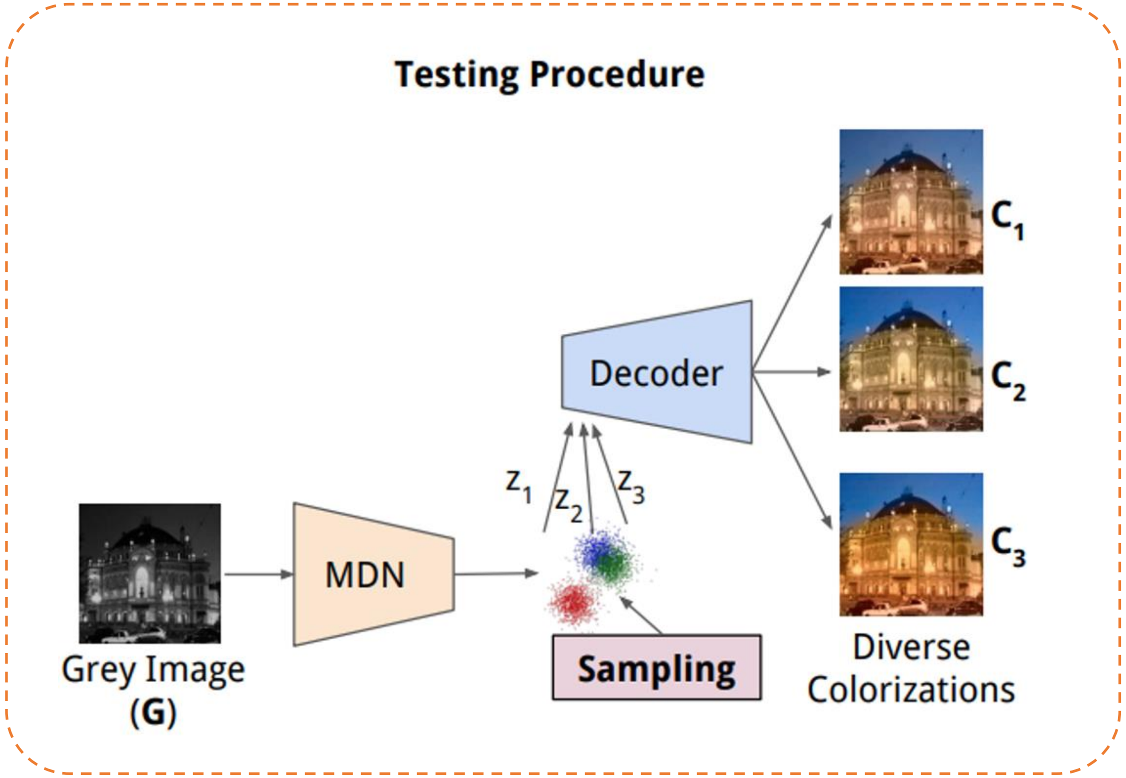
Outline

- VAE Revision
- Image Colorization Overview
- VAE-based Image Colorization Model
- Implementation

Implementation

❖ Review





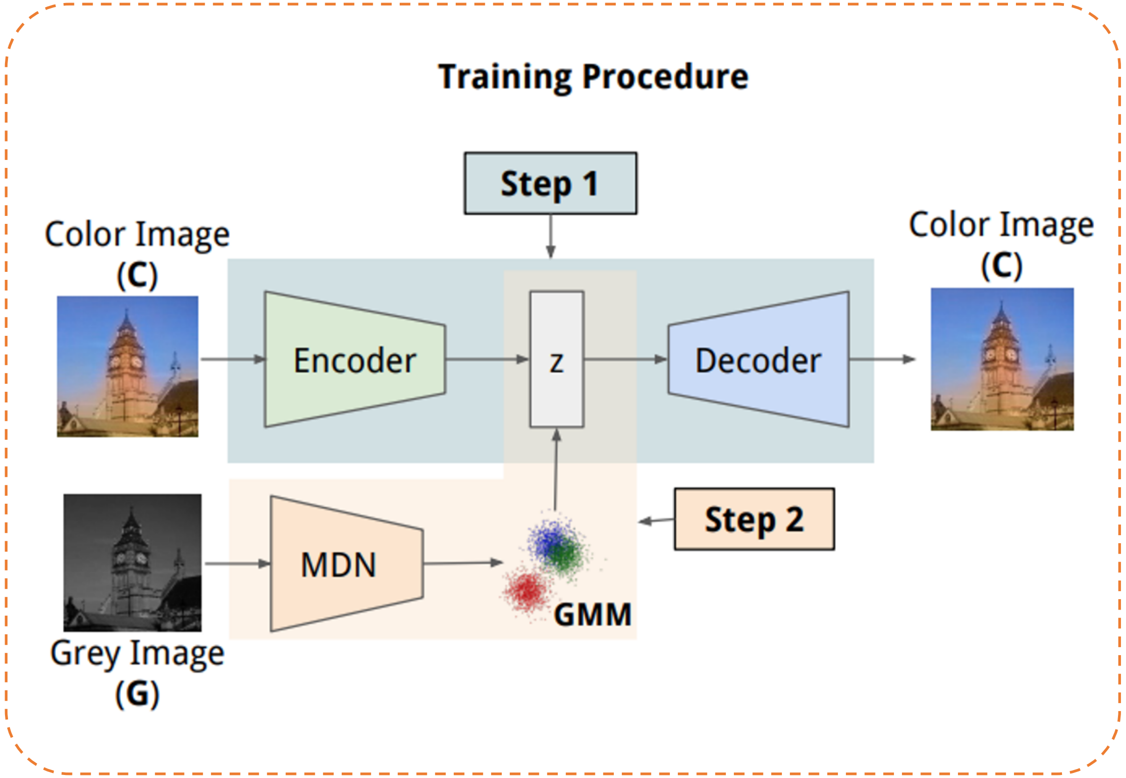
Source: Deshpande, A., Lu, J., Yeh, M. C., Jin Chong, M., & Forsyth, D (2017). **Learning Diverse Image Colorization**. In Proceedings of the IEEE conference on CVPR (pp. 6837-6845).

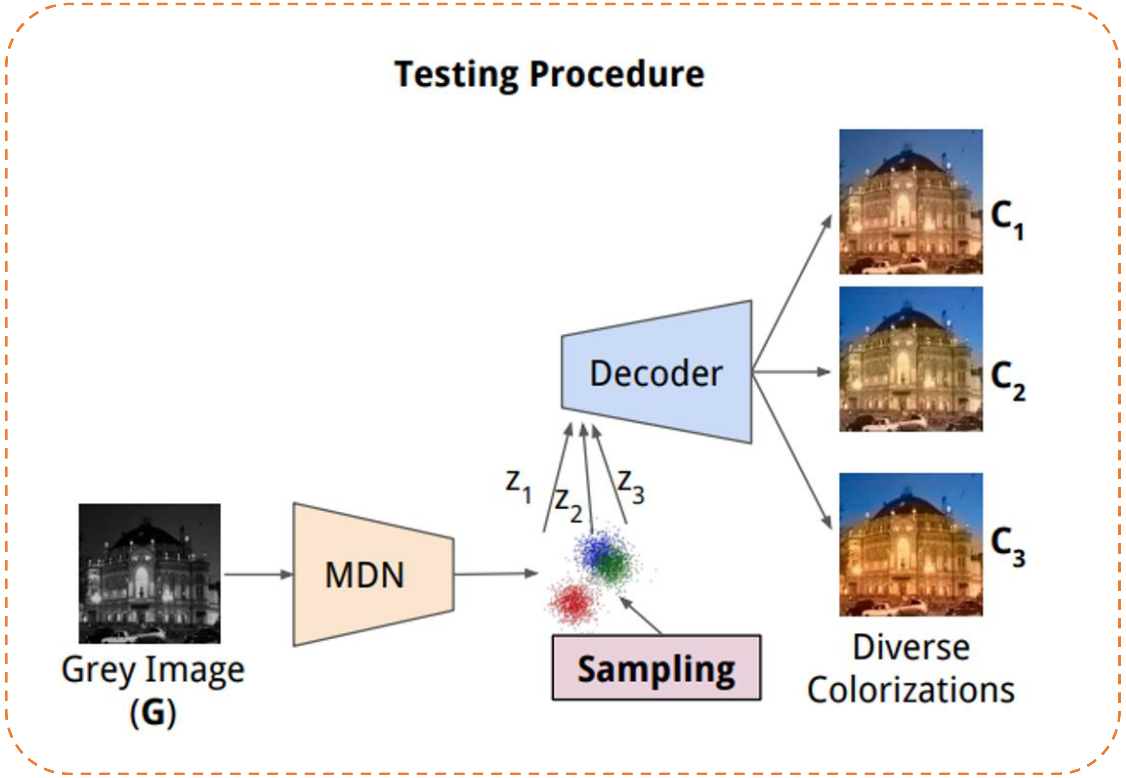
Outline

- VAE Revision
- Image Colorization Overview
- VAE-based Image Colorization Model
- Implementation

Implementation

❖ Review

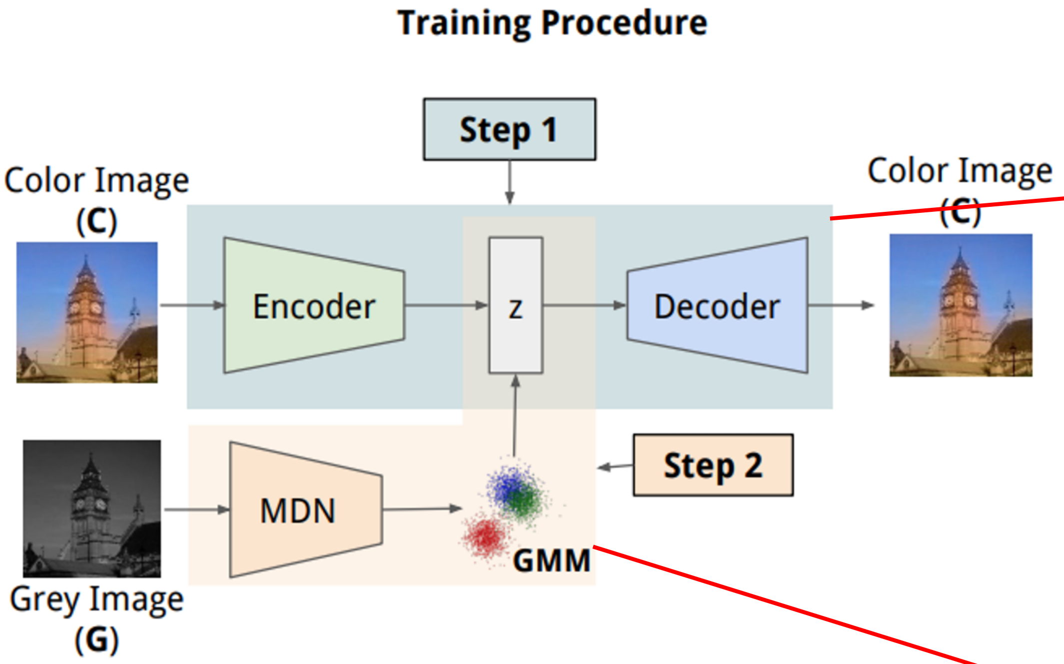


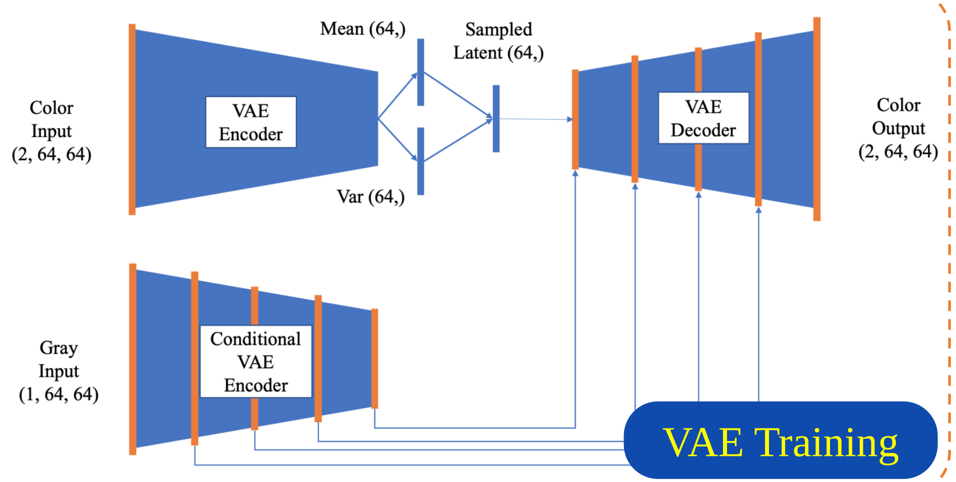


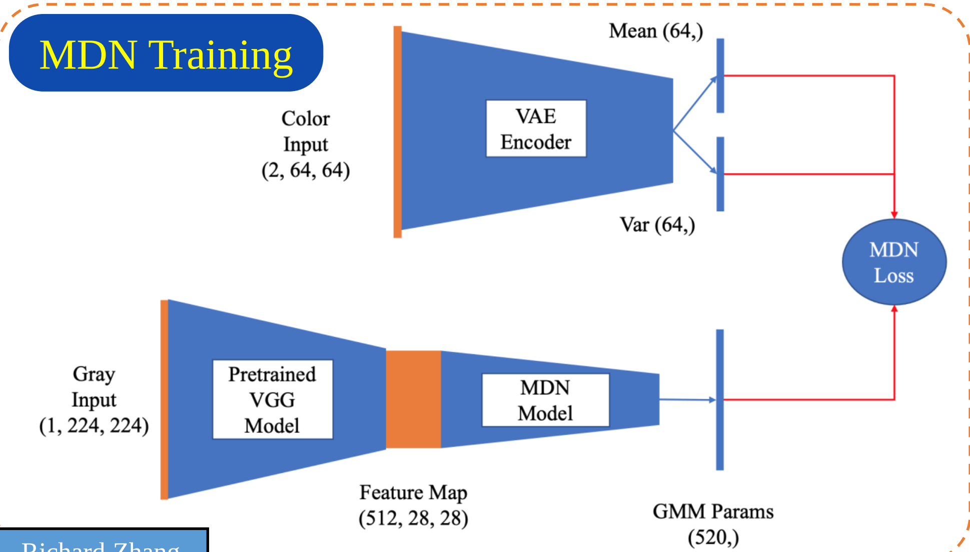
Source: Deshpande, A., Lu, J., Yeh, M. C., Jin Chong, M., & Forsyth, D (2017).
Learning Diverse Image Colorization.
In Proceedings of the IEEE conference on CVPR (pp. 6837-6845).

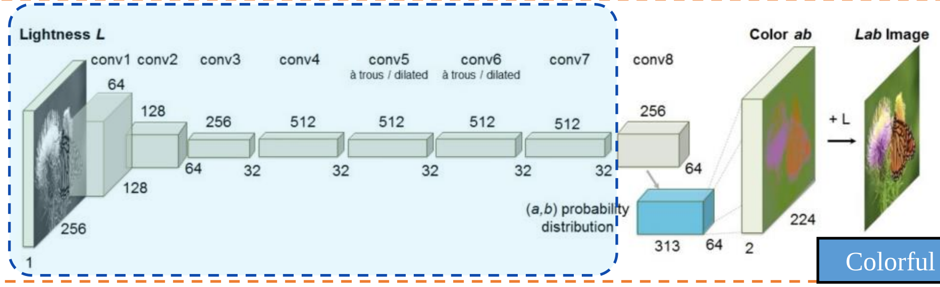
Implementation

❖ Review





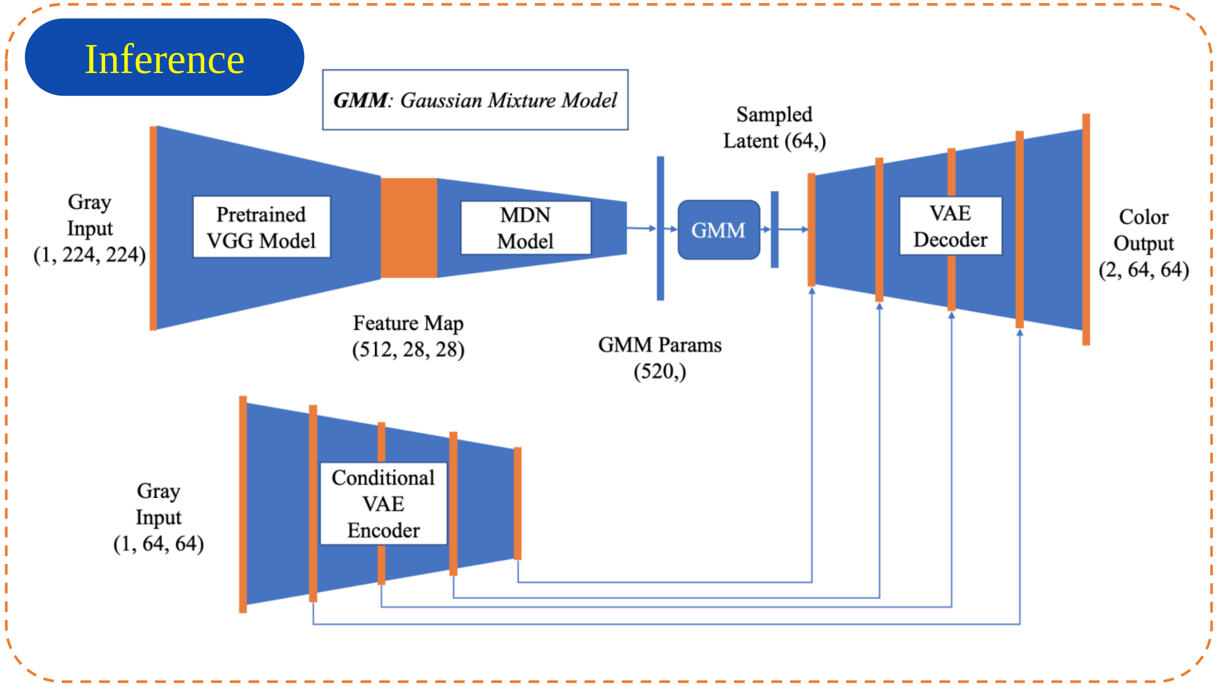


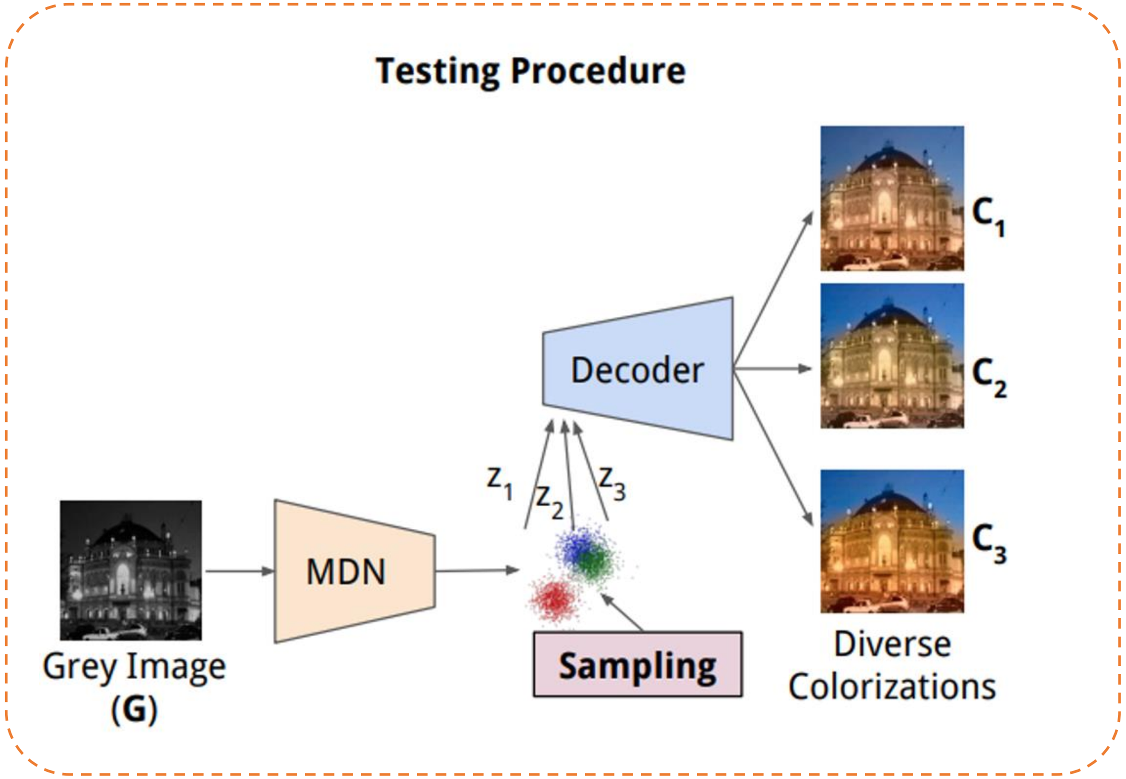


Colorful Image Colorization (2016) – Richard Zhang

Implementation

❖ Review





Implementation

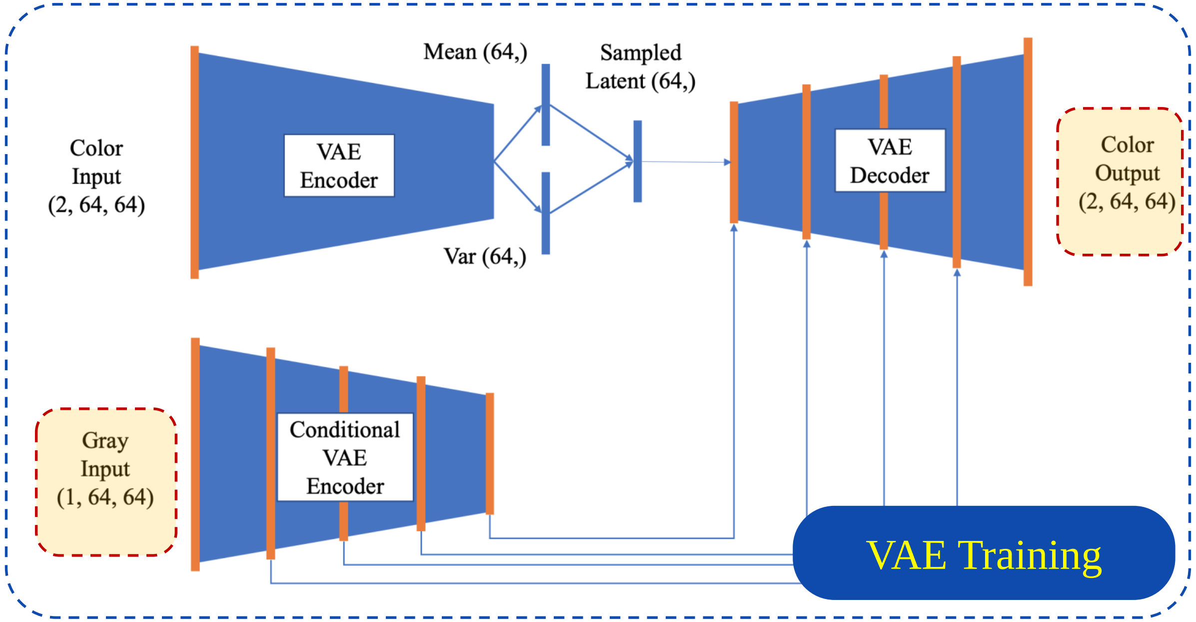
❖ Review

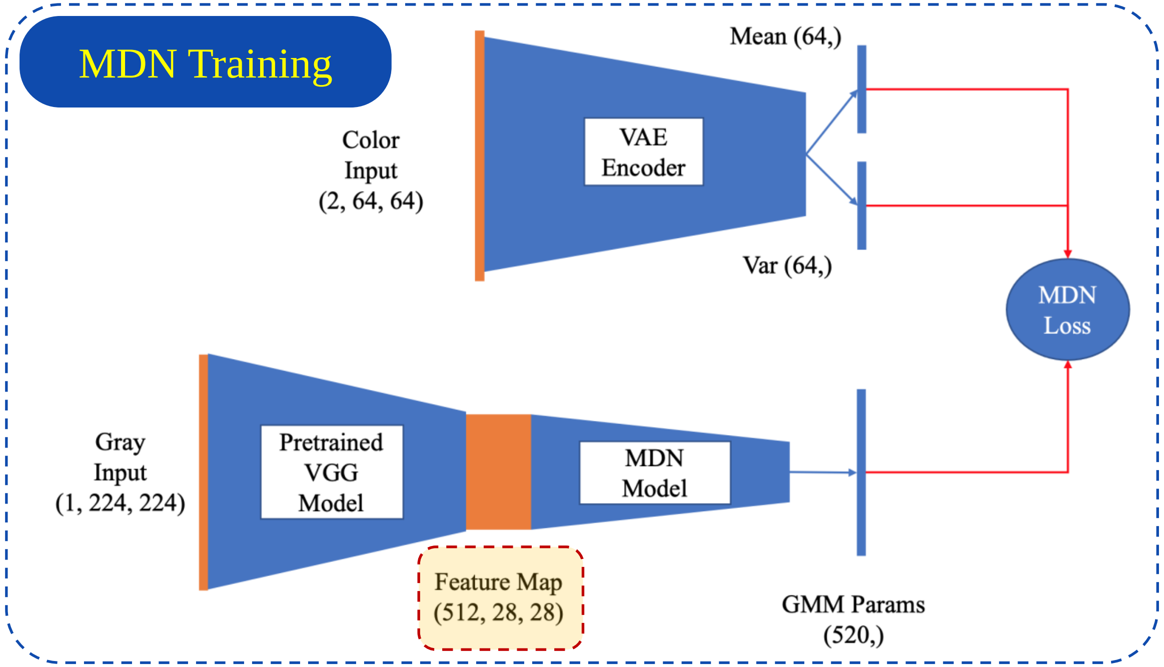
Basic components in an AI project

Data Processing
Model
Loss Function, Optimizer
Trainer
Inference

Implementation

❖ Data





Implementation

❖ Data

```
data
  > featslist
  > imglist
  > lfw_feats
  > lfw_images
  > output
  > zhang_weights
```

```
data
  featslist
    lfw
      list.test.txt
      list.train.txt
  imglist
    lfw
      list.test.vae.txt
      list.train.vae.txt
  > lfw_feats
  > lfw_images
  > output
  > zhang_weights
```

featslist > lfw > list.test.txt

```
data/lfw_feats/Bulent_Ecevit/Bulent_Ecevit_0001.npz
data/lfw_feats/Lee_Hoi-chang/Lee_Hoi-chang_0001.npz
data/lfw_feats/Claire_Hentzen/Claire_Hentzen_0001.npz
```

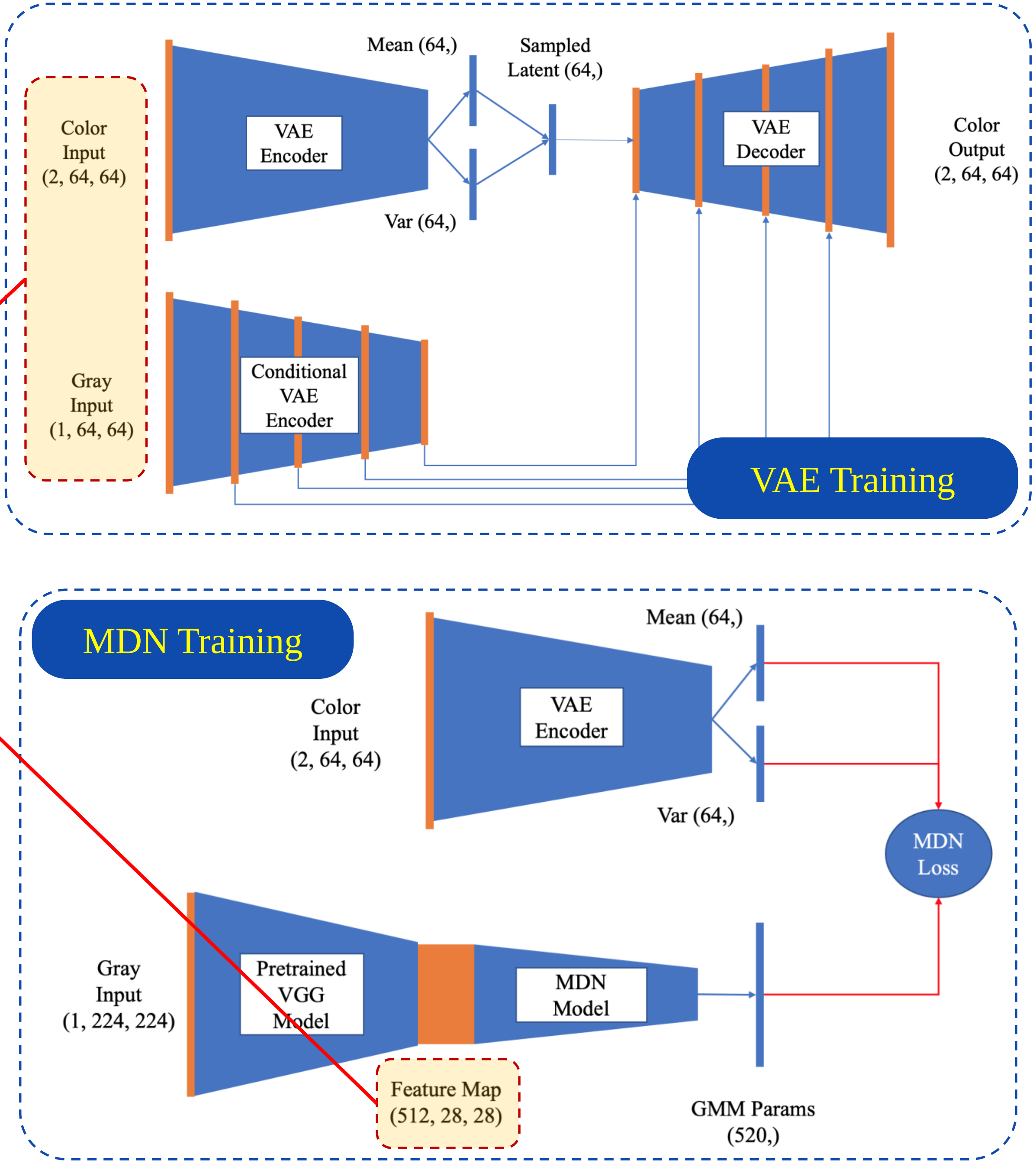
```
lfw_feats
  Aaron_Eckhart
    Aaron_Eckhart_0001.npz
  Aaron_Guiel
    Aaron_Guiel_0001.npz
```

imglist > lfw > list.test.vae.txt

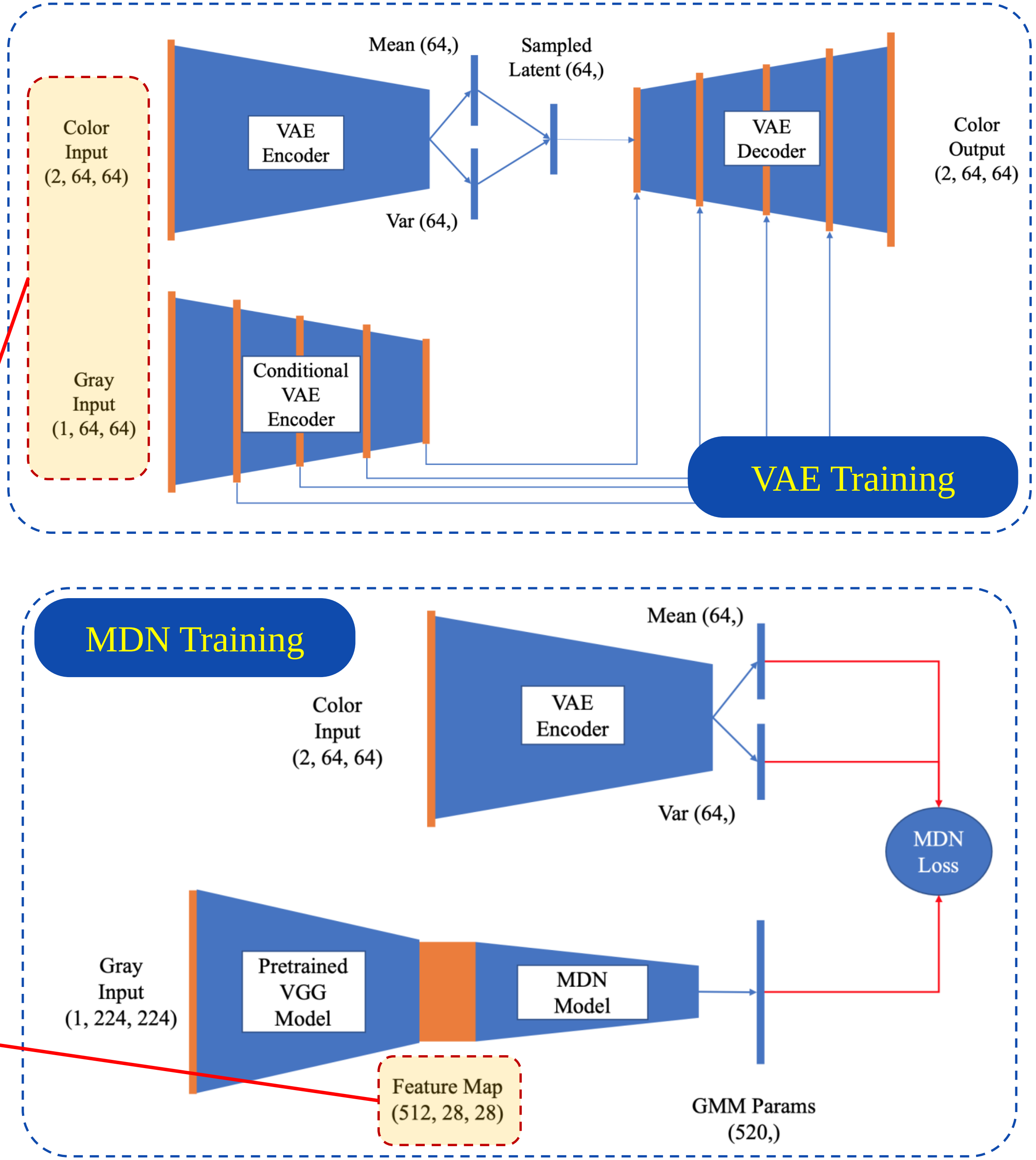
```
data/lfw_images/Bulent_Ecevit/Bulent_Ecevit_0001.jpg
data/lfw_images/Lee_Hoi-chang/Lee_Hoi-chang_0001.jpg
data/lfw_images/Claire_Hentzen/Claire_Hentzen_0001.jpg
```

```
lfw_images
  Aaron_Eckhart
    Aaron_Eckhart_0001.jpg
  Aaron_Guiel
    Aaron_Guiel_0001.jpg
```

```python
class ColorDataset(Dataset):
    def __init__(
        self,
        out_directory,
        listdir=None,
        featslistdir=None,
        shape=(64, 64),
        outshape=(256, 256),
        split="train",
    ):

        # Save paths to a list
        self.img_fns = []
        self.feats_fns = []

        with open("%s/list.%s.vae.txt" % (listdir, split), "r") as ftr:
            for img_fn in ftr:
                self.img_fns.append(img_fn.strip("\n"))

        with open("%s/list.%s.txt" % (featslistdir, split), "r") as ftr:
            for feats_fn in ftr:
                self.feats_fns.append(feats_fn.strip("\n"))

        self.img_num = min(len(self.img_fns), len(self.feats_fns))
        self.shape = shape
        self.outshape = outshape
        self.out_directory = out_directory

        # Create a dictionary to save weight of 313 ab bins
        self.lossweights = None
        countbins = 1.0 / np.load("data/zhang_weights/prior_probs.npy")
        binedges = np.load("data/zhang_weights/ab_quantize.npy").reshape(2, 313)
        lossweights = {}
        for i in range(313):
            if binedges[0, i] not in lossweights:
                lossweights[binedges[0, i]] = {}
            lossweights[binedges[0, i]][binedges[1, i]] = countbins[i]
        self.binedges = binedges
        self.lossweights = lossweights

    def __len__(self):
        return self.img_num
```



```python
def __getitem__(self, idx):
    # Declare empty arrays to get values
    color_ab = np.zeros((2, self.shape[0], self.shape[1]), dtype="f")
    weights = np.ones((2, self.shape[0], self.shape[1]), dtype="f")
    recon_const = np.zeros((1, self.shape[0], self.shape[1]), dtype="f")
    recon_const_outres = np.zeros((1, self.outshape[0], self.outshape[1]), dtype="f")
    greyfeats = np.zeros((512, 28, 28), dtype="f")

    # Read and reshape
    img_large = cv2.imread(self.img_fns[idx])
    if self.shape is not None:
        img = cv2.resize(img_large, (self.shape[0], self.shape[1]))
        img_outres = cv2.resize(img_large, (self.outshape[0], self.outshape[1]))

    # Convert BGR to LAB
    img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    img_lab_outres = cv2.cvtColor(img_outres, cv2.COLOR_BGR2LAB)

    # Normalize to [-1..1]
    img_lab = ((img_lab * 2.0) / 255.0) - 1.0
    img_lab_outres = ((img_lab_outres * 2.0) / 255.0) - 1.0

    recon_const[0, :, :] = img_lab[..., 0]
    recon_const_outres[0, :, :] = img_lab_outres[..., 0]

    color_ab[0, :, :] = img_lab[..., 1].reshape(1, self.shape[0], self.shape[1])
    color_ab[1, :, :] = img_lab[..., 2].reshape(1, self.shape[0], self.shape[1])

    if self.lossweights is not None:
        weights = self.__getweights__(color_ab)

    # Load feature maps
    featobj = np.load(self.feats_fns[idx])
    greyfeats[:, :, :] = featobj["arr_0"]

    return color_ab, recon_const, weights, recon_const_outres, greyfeats
```



Implementation

```python
def __getitem__(self, idx):
    # Declare empty arrays to get values
    color_ab = np.zeros((2, self.shape[0], self.shape[1]), dtype="f")
    weights = np.ones((2, self.shape[0], self.shape[1]), dtype="f")
    recon_const = np.zeros((1, self.shape[0], self.shape[1]), dtype="f")
    recon_const_outres = np.zeros((1, self.outshape[0], self.outshape[1]), dtype="f")
    greyfeats = np.zeros((512, 28, 28), dtype="f")

    # Read and reshape
    img_large = cv2.imread(self.img_fns[idx])
    if self.shape is not None:
        img = cv2.resize(img_large, (self.shape[0], self.shape[1]))
        img_outres = cv2.resize(img_large, (self.outshape[0], self.outshape[1]))

    # Convert BGR to LAB
    img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    img_lab_outres = cv2.cvtColor(img_outres, cv2.COLOR_BGR2LAB)

    # Normalize to [-1..1]
    img_lab = ((img_lab * 2.0) / 255.0) - 1.0
    img_lab_outres = ((img_lab_outres * 2.0) / 255.0) - 1.0

    recon_const[0, :, :] = img_lab[..., 0]
    recon_const_outres[0, :, :] = img_lab_outres[..., 0]

    color_ab[0, :, :] = img_lab[..., 1].reshape(1, self.shape[0], self.shape[1])
    color_ab[1, :, :] = img_lab[..., 2].reshape(1, self.shape[0], self.shape[1])

    if self.lossweights is not None:
        weights = self.__getweights__(color_ab)

    # Load feature maps
    featobj = np.load(self.feats_fns[idx])
    greyfeats[:, :, :] = featobj["arr_0"]

    return color_ab, recon_const, weights, recon_const_outres, greyfeats
```

```python
def __getweights__(self, img):
    """
    Calculate weight values for each pixel of an image.
    """
    img_vec = img.reshape(-1)
    img_vec = img_vec * 128.0
    img_lossweights = np.zeros(img.shape, dtype="f")
    img_vec_a = img_vec[: np.prod(self.shape)]
    binedges_a = self.binedges[0, ...].reshape(-1)
    binid_a = [binedges_a.flat[np.abs(binedges_a - v).argmin()] for v in img_vec_a]
    img_vec_b = img_vec[np.prod(self.shape) :]
    binedges_b = self.binedges[1, ...].reshape(-1)
    binid_b = [binedges_b.flat[np.abs(binedges_b - v).argmin()] for v in img_vec_b]
    binweights = np.array([self.lossweights[v1][v2] for v1, v2 in zip(binid_a, binid_b)])
    img_lossweights[0, :, :] = binweights.reshape(self.shape[0], self.shape[1])
    img_lossweights[1, :, :] = binweights.reshape(self.shape[0], self.shape[1])
    return img_lossweights
```

Implementation

❖ Data: Postprocess

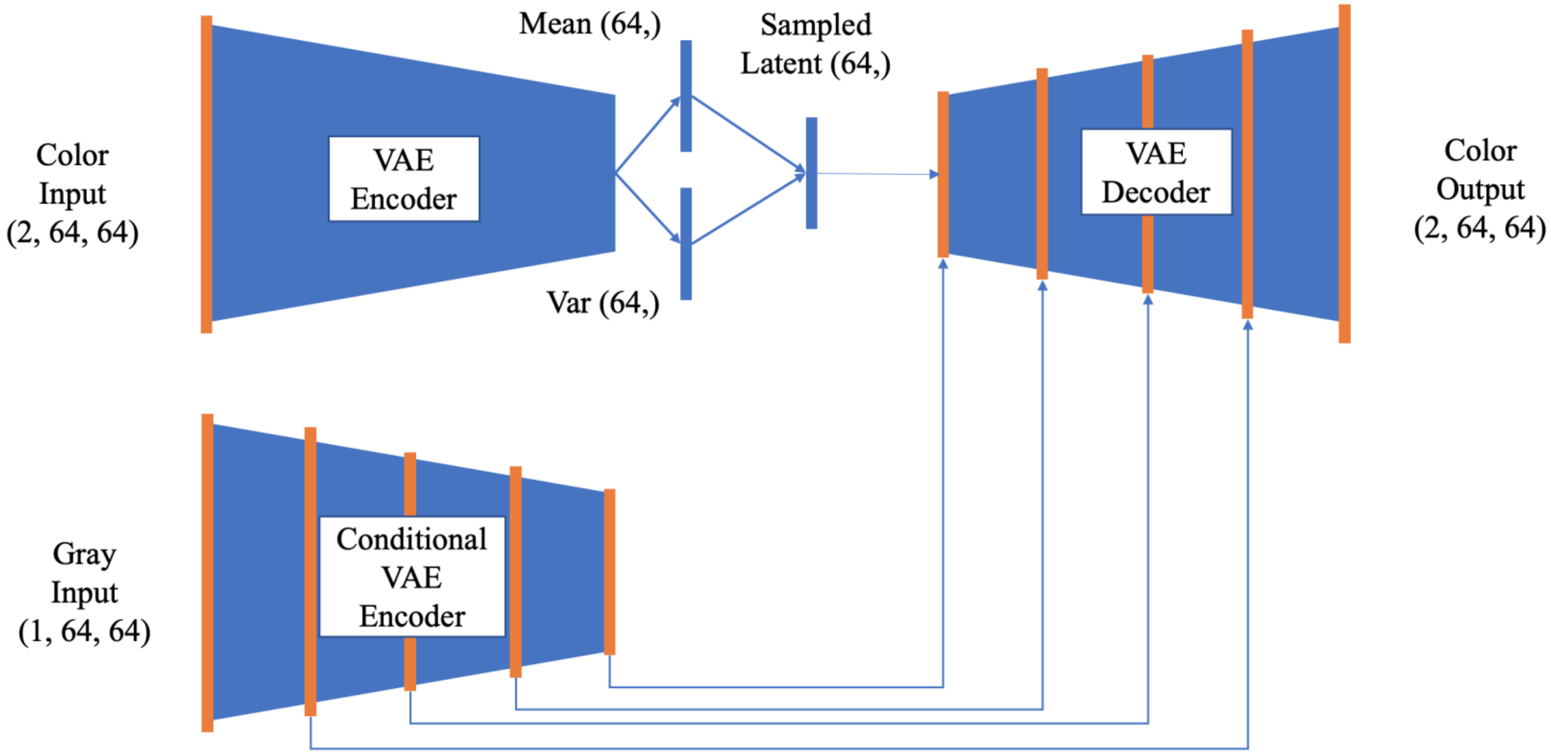
```python
def saveoutput_gt(self, net_op, gt, prefix, batch_size, num_cols=8, net_recon_const=None):
    """
    Save images
    """
    net_out_img = self.__tiledoutput__(net_op, batch_size, num_cols=num_cols, net_recon_const=net_recon_const)
    gt_out_img = self.__tiledoutput__(gt, batch_size, num_cols=num_cols, net_recon_const=net_recon_const)

    num_rows = np.int_(np.ceil((batch_size * 1.0) / num_cols))
    border_img = 255 * np.ones((num_rows * self.outshape[0], 128, 3), dtype="uint8")
    out_fn_pred = "%s/%s.png" % (self.out_directory, prefix)
    cv2.imwrite(out_fn_pred, np.concatenate((net_out_img, border_img, gt_out_img), axis=1))
```

```python
def __decodeimg__(self, img_enc):
    """
    Denormalize from [-1..1] to [0..255]
    """
    img_dec = (((img_enc + 1.0) * 1.0) / 2.0) * 255.0
    img_dec[img_dec < 0.0] = 0.0
    img_dec[img_dec > 255.0] = 255.0
    return cv2.resize(np.uint8(img_dec), (self.outshape[0], self.outshape[1]))
```

```python
def __tiledoutput__(self, net_op, batch_size, num_cols=8, net_recon_const=None):
    """
    Generate a combined image from these inputs by stitching the images into a large image.
    """
    num_rows = np.int_(np.ceil((batch_size * 1.0) / num_cols))
    out_img = np.zeros((num_rows * self.outshape[0], num_cols * self.outshape[1], 3), dtype="uint8")
    img_lab = np.zeros((self.outshape[0], self.outshape[1], 3), dtype="uint8")
    c = 0
    r = 0

    for i in range(batch_size):
        if i % num_cols == 0 and i > 0:
            r = r + 1
            c = 0
        img_lab[..., 0] = self.__decodeimg__(net_recon_const[i, 0, :, :].reshape(self.outshape[0], self.outshape[1]))
        img_lab[..., 1] = self.__decodeimg__(net_op[i, 0, :, :].reshape(self.shape[0], self.shape[1]))
        img_lab[..., 2] = self.__decodeimg__(net_op[i, 1, :, :].reshape(self.shape[0], self.shape[1]))
        img_rgb = cv2.cvtColor(img_lab, cv2.COLOR_LAB2BGR)
        out_img[
            r * self.outshape[0] : (r + 1) * self.outshape[0],
            c * self.outshape[1] : (c + 1) * self.outshape[1],
            ...,
        ] = img_rgb
        c = c + 1

    return out_img
```

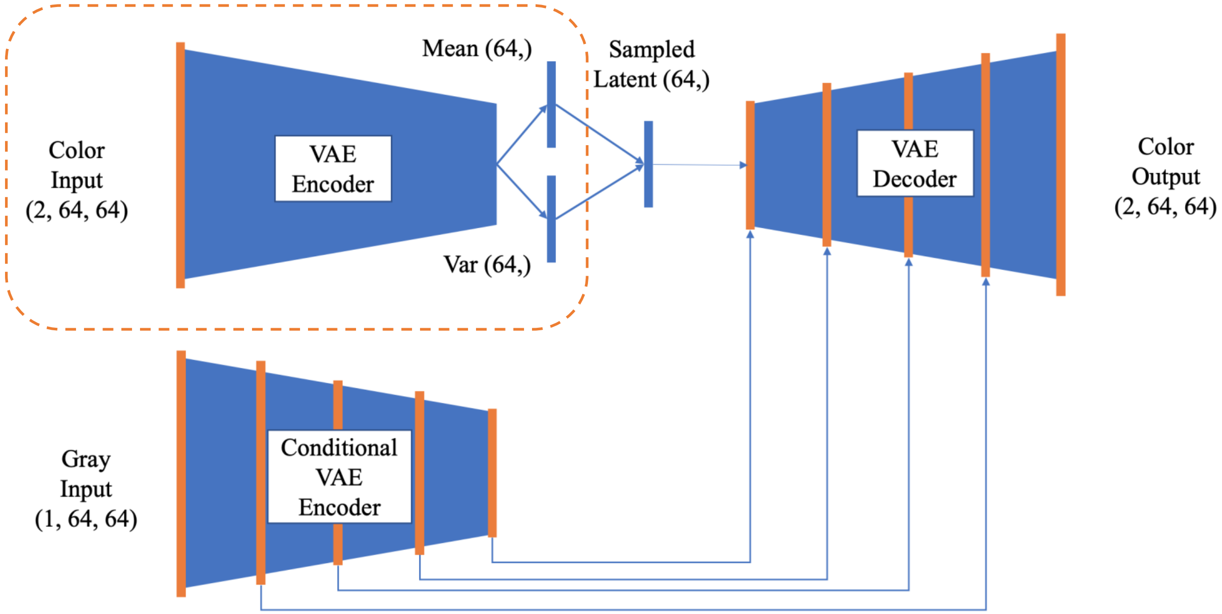
Implementation

❖ Models: CVAE



Implementation

❖ Models: CVAE – Encoder Block

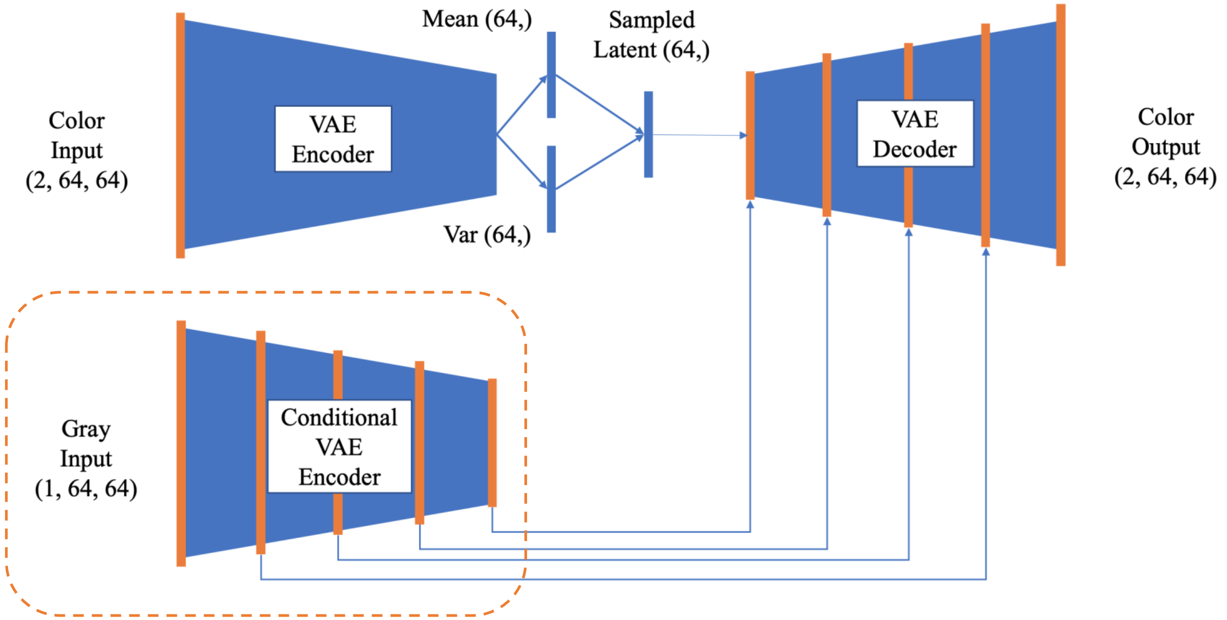



```python
# Encoder layers
self.enc_conv1 = nn.Conv2d(2, 128, 5, stride=2, padding=2)
self.enc_bn1 = nn.BatchNorm2d(128)
self.enc_conv2 = nn.Conv2d(128, 256, 5, stride=2, padding=2)
self.enc_bn2 = nn.BatchNorm2d(256)
self.enc_conv3 = nn.Conv2d(256, 512, 5, stride=2, padding=2)
self.enc_bn3 = nn.BatchNorm2d(512)
self.enc_conv4 = nn.Conv2d(512, 1024, 3, stride=2, padding=1)
self.enc_bn4 = nn.BatchNorm2d(1024)
self.enc_fc1 = nn.Linear(4*4*1024, self.hidden_size*2)
self.enc_dropout1 = nn.Dropout(p=0.7)
```

```python
def encoder(self, x):                       # (2, 64, 64)
    x = F.relu(self.enc_conv1(x))
    x = self.enc_bn1(x)                     # (128, 32, 32)
    x = F.relu(self.enc_conv2(x))
    x = self.enc_bn2(x)                     # (256, 16, 16)
    x = F.relu(self.enc_conv3(x))
    x = self.enc_bn3(x)                     # (512, 8, 8)
    x = F.relu(self.enc_conv4(x))
    x = self.enc_bn4(x)                     # (1024, 4, 4)
    x = x.view(-1, 4*4*1024)
    x = self.enc_dropout1(x)
    x = self.enc_fc1(x)                     # (128,)
    mu = x[..., :self.hidden_size]          # (64,)
    logvar = x[..., self.hidden_size:]      # (64,)
    return mu, logvar
```

Implementation

❖ Models: CVAE – Conditional Encoder Block

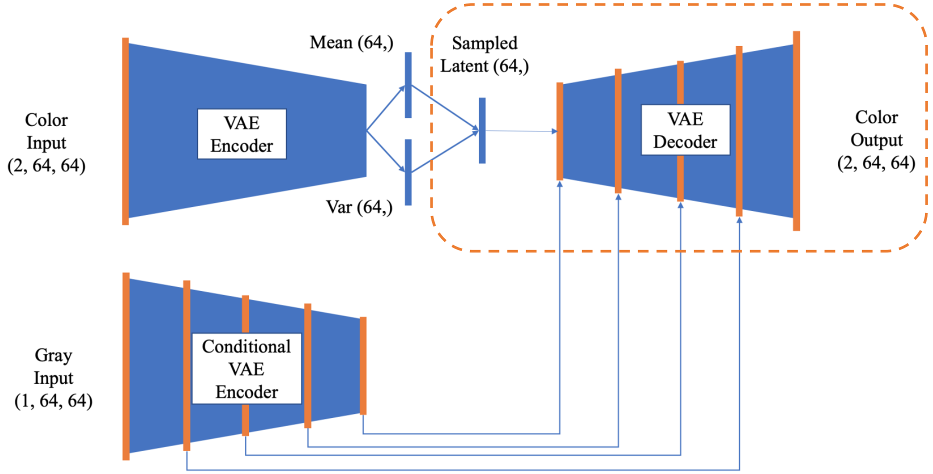



```python
# Conditional encoder layers
self.cond_enc_conv1 = nn.Conv2d(1, 128, 5, stride=2, padding=2)
self.cond_enc_bn1 = nn.BatchNorm2d(128)
self.cond_enc_conv2 = nn.Conv2d(128, 256, 5, stride=2, padding=2)
self.cond_enc_bn2 = nn.BatchNorm2d(256)
self.cond_enc_conv3 = nn.Conv2d(256, 512, 5, stride=2, padding=2)
self.cond_enc_bn3 = nn.BatchNorm2d(512)
self.cond_enc_conv4 = nn.Conv2d(512, 1024, 3, stride=2, padding=1)
self.cond_enc_bn4 = nn.BatchNorm2d(1024)
```

```python
def cond_encoder(self, x):                              # (1, 64, 64)
    x = F.relu(self.cond_enc_conv1(x))
    sc_feat32 = self.cond_enc_bn1(x)                    # (128, 32, 32)
    x = F.relu(self.cond_enc_conv2(sc_feat32))
    sc_feat16 = self.cond_enc_bn2(x)                    # (256, 16, 16)
    x = F.relu(self.cond_enc_conv3(sc_feat16))
    sc_feat8 = self.cond_enc_bn3(x)                     # (512, 8, 8)
    x = F.relu(self.cond_enc_conv4(sc_feat8))
    sc_feat4 = self.cond_enc_bn4(x)                     # (1024, 4, 4)
    return sc_feat32, sc_feat16, sc_feat8, sc_feat4
```

Implementation

❖ Models: CVAE – Decoder Block

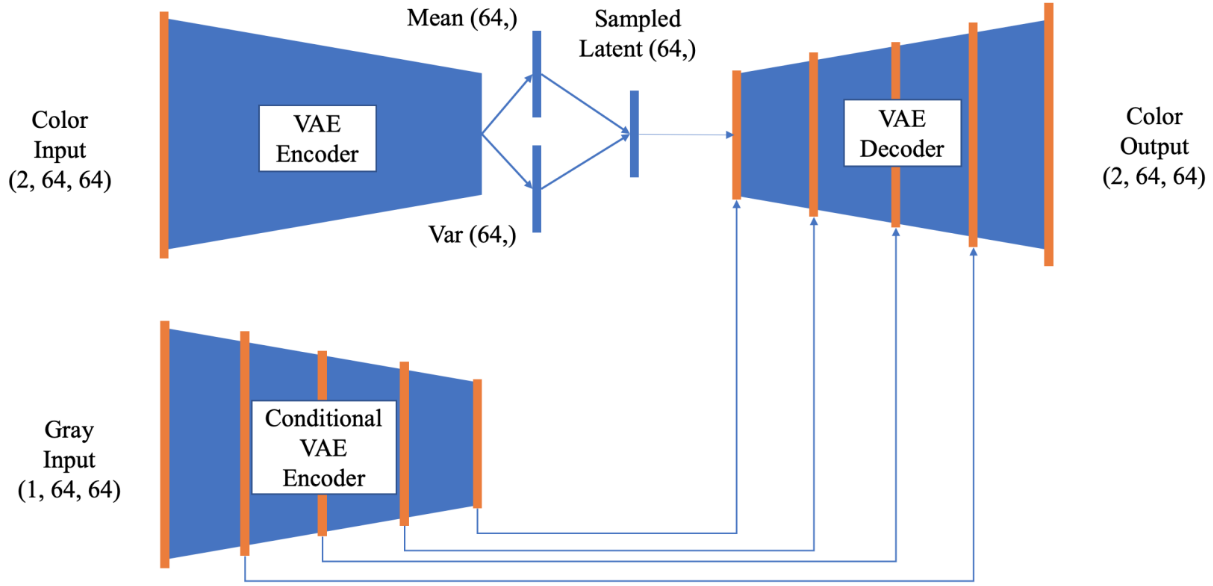



```python
# Decoder layers
self.dec_upsamp1 = nn.Upsample(scale_factor=4, mode='bilinear')
self.dec_conv1 = nn.Conv2d(1024+self.hidden_size, 512, 3, stride=1, padding=1)
self.dec_bn1 = nn.BatchNorm2d(512)
self.dec_upsamp2 = nn.Upsample(scale_factor=2, mode='bilinear')
self.dec_conv2 = nn.Conv2d(512*2, 256, 5, stride=1, padding=2)
self.dec_bn2 = nn.BatchNorm2d(256)
self.dec_upsamp3 = nn.Upsample(scale_factor=2, mode='bilinear')
self.dec_conv3 = nn.Conv2d(256*2, 128, 5, stride=1, padding=2)
self.dec_bn3 = nn.BatchNorm2d(128)
self.dec_upsamp4 = nn.Upsample(scale_factor=2, mode='bilinear')
self.dec_conv4 = nn.Conv2d(128*2, 64, 5, stride=1, padding=2)
self.dec_bn4 = nn.BatchNorm2d(64)
self.dec_upsamp5 = nn.Upsample(scale_factor=2, mode='bilinear')
self.dec_conv5 = nn.Conv2d(64, 2, 5, stride=1, padding=2)
```

```python
def decoder(self, z, sc_feat32, sc_feat16, sc_feat8, sc_feat4):
    x = z.view(-1, self.hidden_size, 1, 1)        # (64, 1, 1)
    x = self.dec_upsamp1(x)                       # (64, 4, 4)
    x = torch.cat([x, sc_feat4], 1)               # (64+1024, 4, 4)
    x = F.relu(self.dec_conv1(x))                 # (512, 4, 4)
    x = self.dec_bn1(x)                           # (512, 4, 4)
    x = self.dec_upsamp2(x)                       # (512, 8, 8)
    x = torch.cat([x, sc_feat8], 1)               # (512+512, 8, 8)
    x = F.relu(self.dec_conv2(x))                 # (256, 8, 8)
    x = self.dec_bn2(x)                           # (256, 8, 8)
    x = self.dec_upsamp3(x)                       # (256, 16, 16)
    x = torch.cat([x, sc_feat16], 1)              # (256+256, 16, 16)
    x = F.relu(self.dec_conv3(x))                 # (128, 16, 16)
    x = self.dec_bn3(x)                           # (128, 16, 16)
    x = self.dec_upsamp4(x)                       # (128, 32, 32)
    x = torch.cat([x, sc_feat32], 1)              # (128+128, 32, 32)
    x = F.relu(self.dec_conv4(x))                 # (64, 32, 32)
    x = self.dec_bn4(x)                           # (64, 32, 32)
    x = self.dec_upsamp5(x)                       # (64, 64, 64)
    x = torch.tanh(self.dec_conv5(x))             # (2, 64, 64)
    return x
```

Implementation

❖ Models: CVAE – Forward

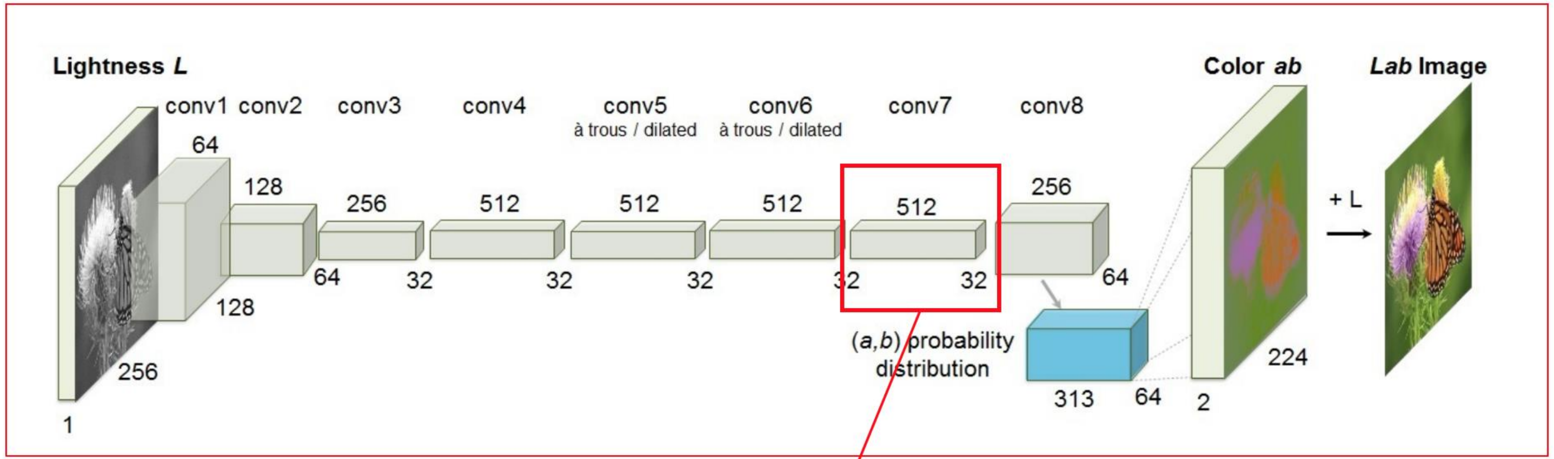


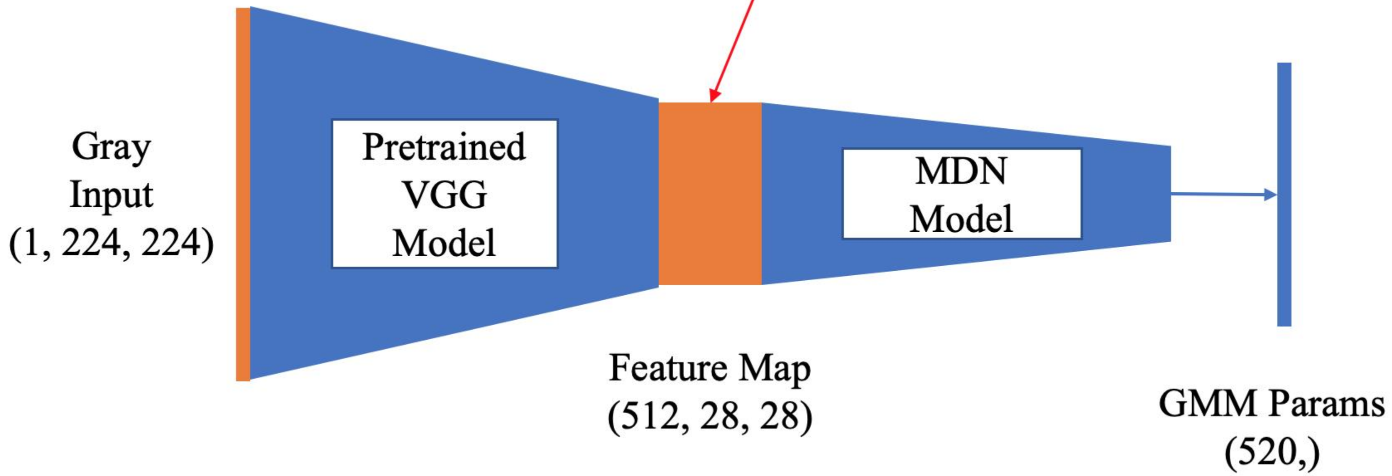

```python
def forward(self, color, greylevel, z_in=None):
    sc_feat32, sc_feat16, sc_feat8, sc_feat4 = self.cond_encoder(greylevel)
    mu, logvar = self.encoder(color)
    if self.training:
        stddev = torch.sqrt(torch.exp(logvar))
        eps = torch.randn_like(stddev)
        z = mu + eps * stddev
        z = z.to(greylevel.device)
    else:
        z = z_in
        z = z.to(greylevel.device)
    color_out = self.decoder(z, sc_feat32, sc_feat16, sc_feat8, sc_feat4)
    return mu, logvar, color_out
```

Implementation

❖ Models: Mixture Density Network

Note: The MDN Model leverage the feature map extracted from a pretrained VGG model.

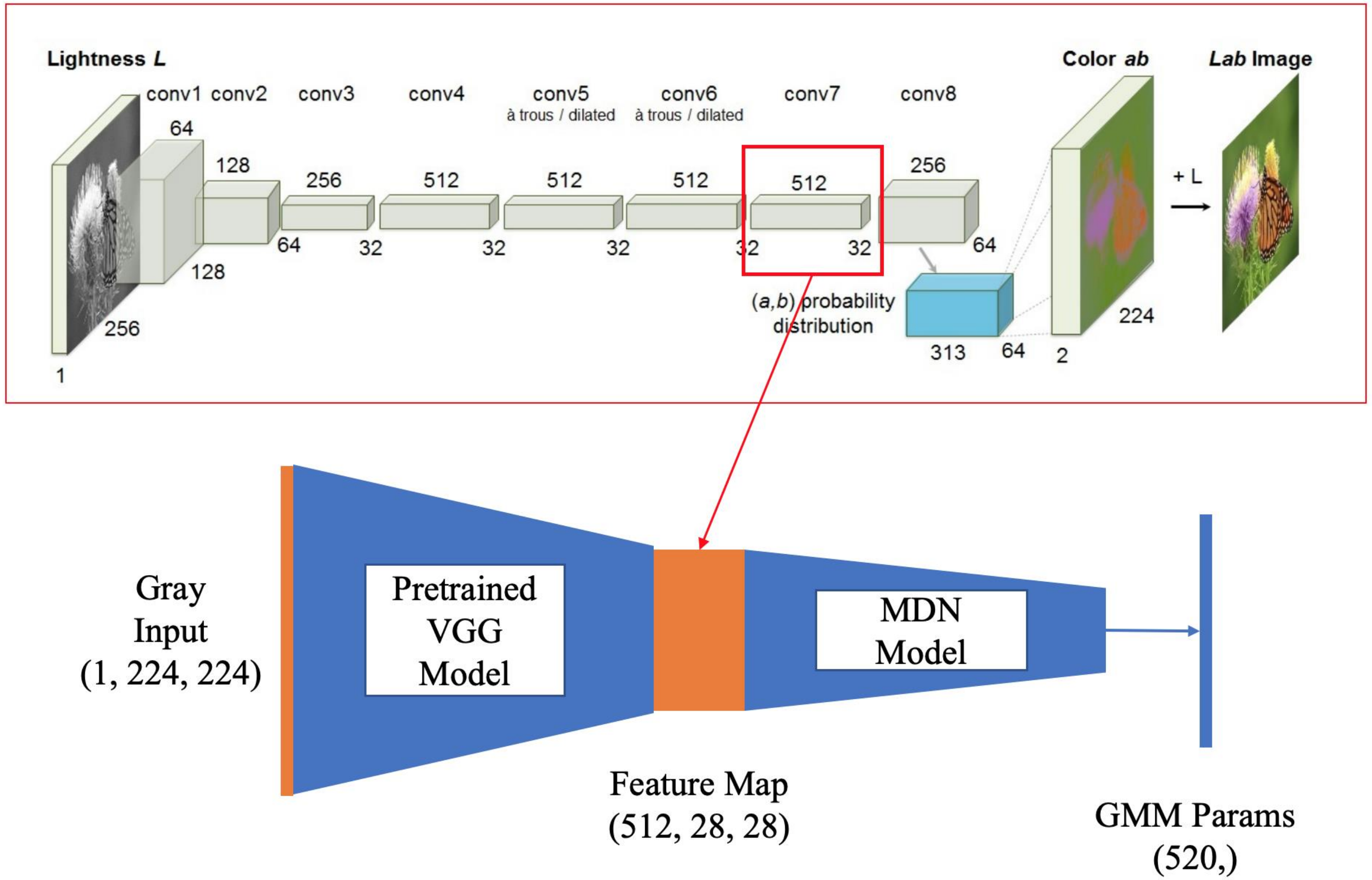




Implementation

❖ Models: Mixture Density Network

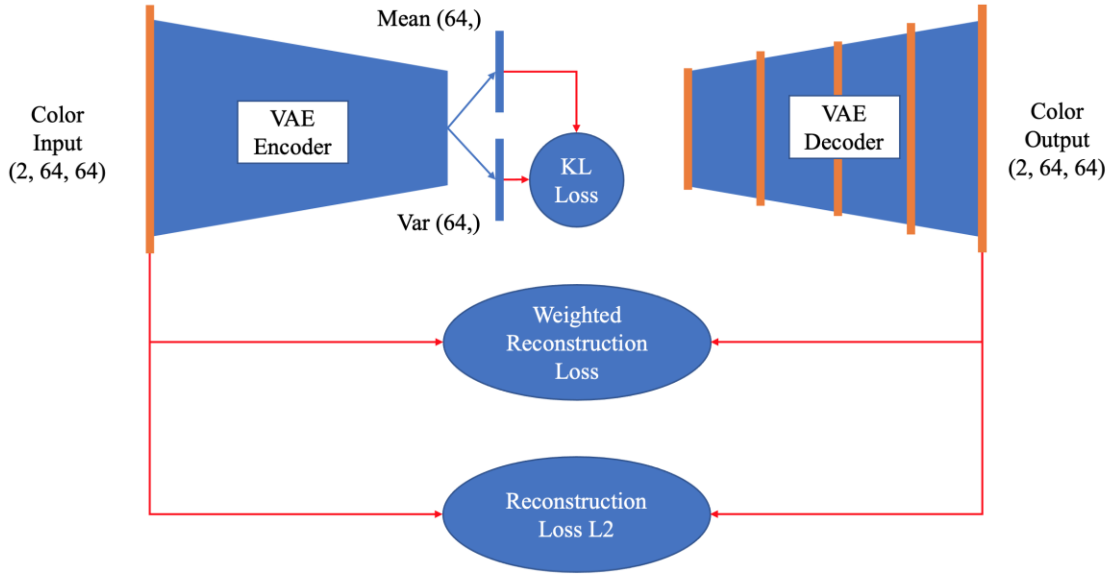
Note: The MDN Model leverage the feature map extracted from a pretrained VGG model.




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

        self.feats_nch = 512
        self.hidden_size = 64
        self.nmix = 8
        self.nout = (self.hidden_size + 1) * self.nmix

        # Define MDN Layers - (512, 64, 64)
        self.model = nn.Sequential(
            nn.Conv2d(self.feats_nch, 384, 5, stride=1, padding=2), # (384, 28, 28)
            nn.BatchNorm2d(384),
            nn.ReLU(),
            nn.Conv2d(384, 320, 5, stride=1, padding=2),            # (320, 28, 28)
            nn.BatchNorm2d(320),
            nn.ReLU(),
            nn.Conv2d(320, 288, 5, stride=1, padding=2),            # (288, 28, 28)
            nn.BatchNorm2d(288),
            nn.ReLU(),
            nn.Conv2d(288, 256, 5, stride=2, padding=2),            # (256, 14, 14)
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 128, 5, stride=1, padding=2),            # (128, 14, 14)
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 96, 5, stride=2, padding=2),             # (96, 7, 7)
            nn.BatchNorm2d(96),
            nn.ReLU(),
            nn.Conv2d(96, 64, 5, stride=2, padding=2),              # (64, 4, 4)
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Dropout(p=0.7)
        )

        self.fc = nn.Linear(4 * 4 * 64, self.nout)

    def forward(self, feats):
        x = self.model(feats)
        x = x.view(-1, 4 * 4 * 64)
        x = F.relu(x)
        x = F.dropout(x, p=0.7, training=self.training)
        x = self.fc(x)
        return x
```

Implementation

❖ Loss Function: VAE Loss

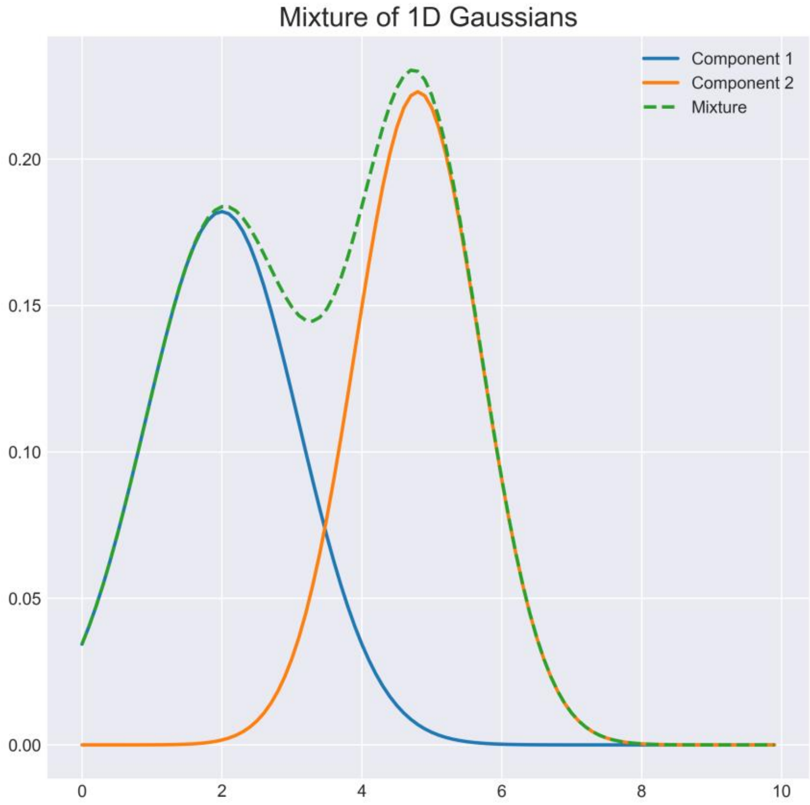



```python
def vae_loss(mu, logvar, pred, gt, lossweights, batchsize):
    '''
    Return the loss values of the VAE model.
    '''
    kl_element = torch.add(torch.add(torch.add(mu.pow(2), logvar.exp()), -1), logvar.mul(-1))
    kl_loss = torch.sum(kl_element).mul(0.5)
    gt = gt.reshape(-1, 64 * 64 * 2)
    pred = pred.reshape(-1, 64 * 64 * 2)
    recon_element = torch.sqrt(torch.sum(torch.mul(torch.add(gt, pred.mul(-1)).pow(2), lossweights), 1))
    recon_loss = torch.sum(recon_element).mul(1.0 / (batchsize))

    recon_element_l2 = torch.sqrt(torch.sum(torch.add(gt, pred.mul(-1)).pow(2), 1))
    recon_loss_l2 = torch.sum(recon_element_l2).mul(1.0 / (batchsize))

    return kl_loss, recon_loss, recon_loss_l2
```

Implementation

❖ Gaussian Mixture Models (GMM)

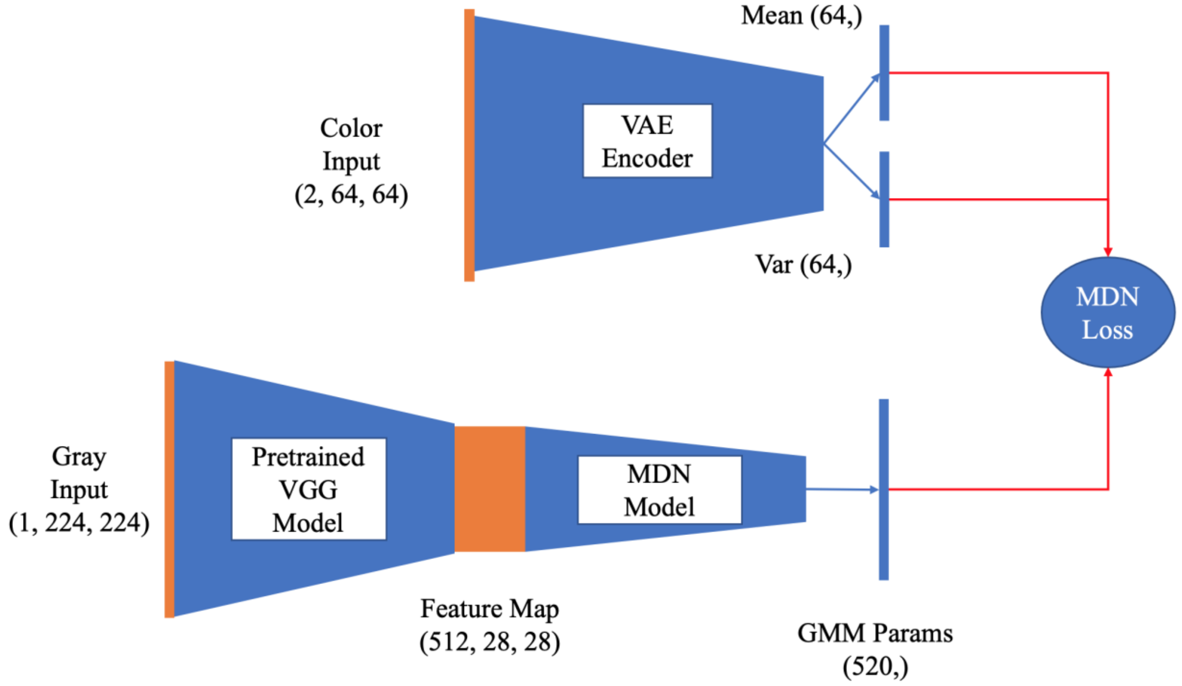
- A mean μ: center of the distribution.
- A covariance Σ: width of the distribution.
- A mixing probability π: magnitude of the distribution



Source: angusturner.github.io/generative_models

Implementation

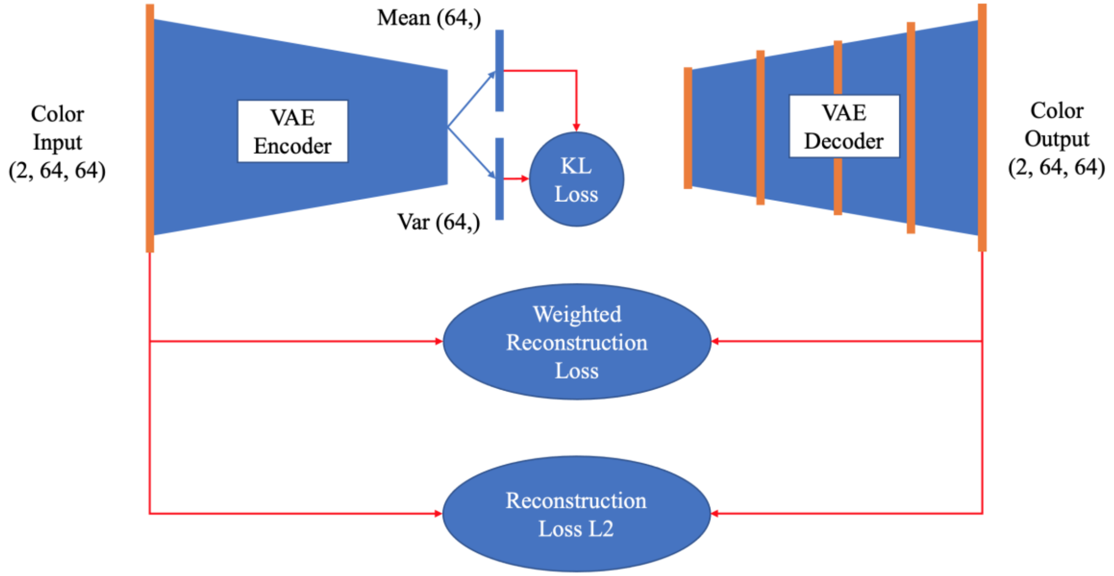
❖ Loss Function: MDN Loss




```python
def get_gmm_coeffs(gmm_params):
    """
    Return the distribution coefficients of the GMM.
    """
    gmm_mu = gmm_params[..., : args["hiddensize"] * args["nmix"]]
    gmm_mu.contiguous()
    gmm_pi_activ = gmm_params[..., args["hiddensize"] * args["nmix"] :]
    gmm_pi_activ.contiguous()
    gmm_pi = F.softmax(gmm_pi_activ, dim=1)
    return gmm_mu, gmm_pi

def mdn_loss(gmm_params, mu, stddev, batchsize):
    """
    Calculates the loss by comparing two distribution
    - the predicted distribution of the MDN (given by gmm_mu and gmm_pi) with
    - the target distribution created by the Encoder block (given by mu and stddev).
    """
    gmm_mu, gmm_pi = get_gmm_coeffs(gmm_params)
    eps = torch.randn(stddev.size()).normal_().cuda()
    z = torch.add(mu, torch.mul(eps, stddev))
    z_flat = z.repeat(1, args["nmix"])
    z_flat = z_flat.reshape(batchsize * args["nmix"], args["hiddensize"])
    gmm_mu_flat = gmm_mu.reshape(batchsize * args["nmix"], args["hiddensize"])
    dist_all = torch.sqrt(torch.sum(torch.add(z_flat, gmm_mu_flat.mul(-1)).pow(2).mul(50), 1))
    dist_all = dist_all.reshape(batchsize, args["nmix"])
    dist_min, selectids = torch.min(dist_all, 1)
    gmm_pi_min = torch.gather(gmm_pi, 1, selectids.reshape(-1, 1))
    gmm_loss = torch.mean(torch.add(-1 * torch.log(gmm_pi_min + 1e-30), dist_min))
    gmm_loss_l2 = torch.mean(dist_min)
    return gmm_loss, gmm_loss_l2
```

Implementation

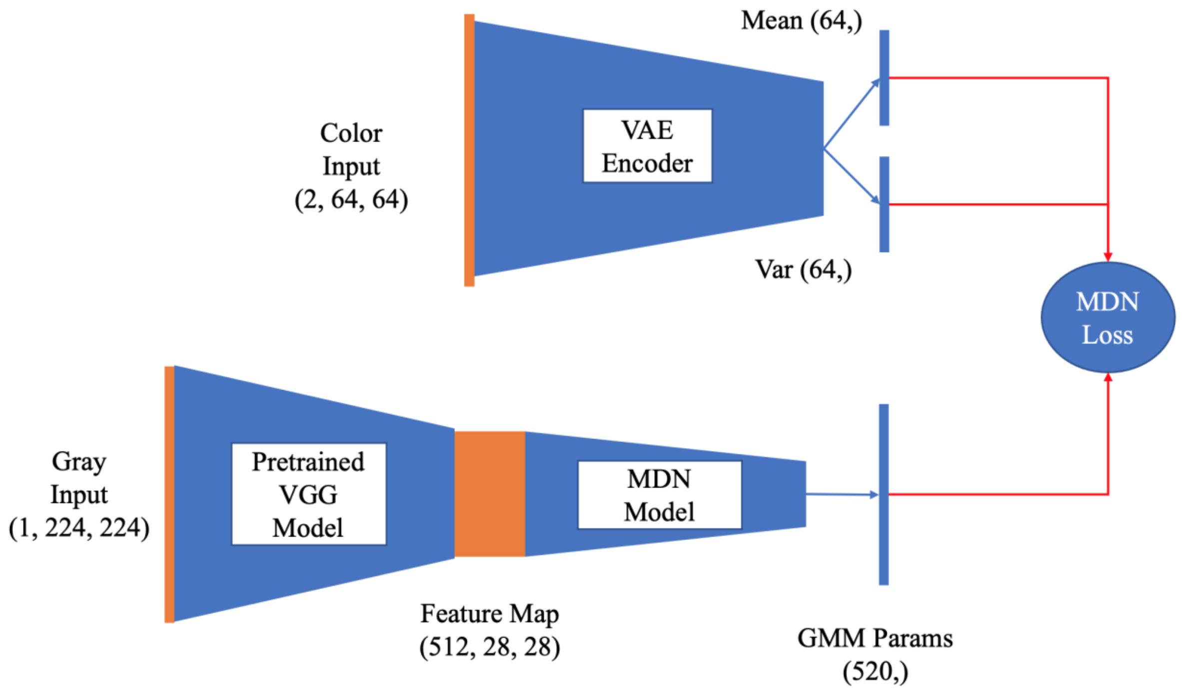
❖ Training: Train CVAE




```python
for epochs in range(nepochs):
    train_loss = 0.0

    for batch_idx, (
        batch,
        batch_recon_const,
        batch_weights,
        batch_recon_const_outres,
        _,
    ) in tqdm(enumerate(data_loader), total=nbatches):

        input_color = batch.cuda()
        lossweights = batch_weights.cuda()
        lossweights = lossweights.reshape(batchsize, -1)
        input_greylevel = batch_recon_const.cuda()
        z = torch.randn(batchsize, hiddensize)

        optimizer.zero_grad()
        mu, logvar, color_out = model(input_color, input_greylevel, z)
        kl_loss, recon_loss, recon_loss_l2 = vae_loss(mu, logvar, color_out, input_color, lossweights, batchsize)
        loss = kl_loss.mul(1e-2) + recon_loss
        recon_loss_l2.detach()
        loss.backward()
        optimizer.step()

        train_loss = train_loss + recon_loss_l2.item()

        if batch_idx % args["logstep"] == 0:
            data.saveoutput_gt(
                color_out.cpu().data.numpy(),
                batch.numpy(),
                "train_%05d_%05d" % (epochs, batch_idx),
                batchsize,
                net_recon_const=batch_recon_const_outres.numpy()
            )

    train_loss = (train_loss * 1.0) / (nbatches)
    test_loss = test_vae(model)
    print(f"End of epoch {epochs:3d} | Train Loss {train_loss:8.3f} | Test Loss {test_loss:8.3f} ")

    # Save VAE model
    torch.save(model.state_dict(), "%s/models/model_vae.pth" % (out_dir))
```

Implementation

❖ Training: Train MDN

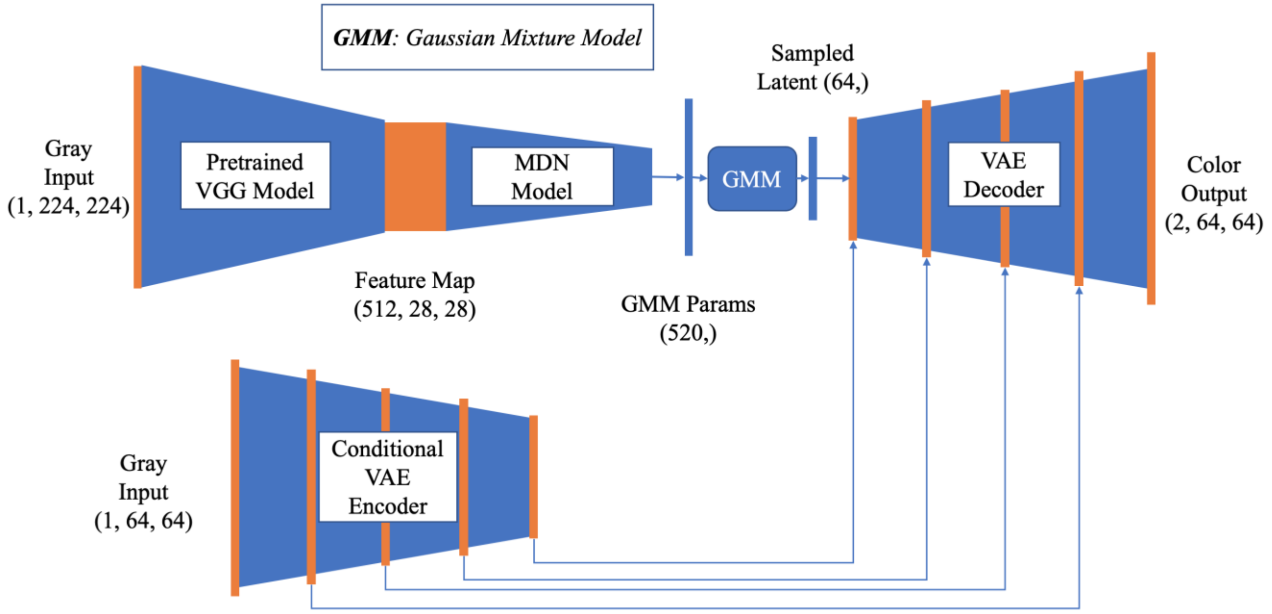



```python
model_vae.eval()
model_mdn.train()

# Train
itr_idx = 0
for epochs_mdn in range(nepochs):
    train_loss = 0.0

    for batch_idx, (
        batch,
        batch_recon_const,
        batch_weights,
        _,
        batch_feats,
    ) in tqdm(enumerate(data_loader), total=nbatches):

        input_color = batch.cuda()
        input_greylevel = batch_recon_const.cuda()
        input_feats = batch_feats.cuda()
        z = torch.randn(batchsize, hiddensize)
        optimizer.zero_grad()

        # Get the parameters of the posterior distribution
        mu, logvar, _ = model_vae(input_color, input_greylevel, z)

        # Get the GMM vector
        mdn_gmm_params = model_mdn(input_feats)

        # Compare 2 distributions
        loss, loss_l2 = mdn_loss(mdn_gmm_params, mu, torch.sqrt(torch.exp(logvar)), batchsize)

        loss.backward()
        optimizer.step()
        train_loss = train_loss + loss.item()

    train_loss = (train_loss * 1.0) / (nbatches)
    test_loss = test_mdn(model_vae, model_mdn)
    print(f"End of epoch {epochs_mdn:3d} | Train Loss {train_loss:8.3f} |  Test Loss {test_loss:8.3f}")

    # Save MDN model
    torch.save(model_mdn.state_dict(), "%s/models_mdn/model_mdn.pth" % (out_dir))
```

AI VIETNAM
All-in-One Course

Implementation

❖ Training: Inference



```python
# Infer
for batch_idx, (
    batch,
    batch_recon_const,
    batch_weights,
    batch_recon_const_outres,
    batch_feats,
) in tqdm(enumerate(data_loader), total=nbatches):

    input_feats = batch_feats.cuda()

    # Get GMM parameters
    mdn_gmm_params = model_mdn(input_feats)
    gmm_mu, gmm_pi = get_gmm_coeffs(mdn_gmm_params)
    gmm_pi = gmm_pi.reshape(-1, 1)
    gmm_mu = gmm_mu.reshape(-1, hiddensize)

    for j in range(batchsize):
        batch_j = np.tile(batch[j, ...].numpy(), (batchsize, 1, 1, 1))
        batch_recon_const_j = np.tile(batch_recon_const[j, ...].numpy(), (batchsize, 1, 1, 1))
        batch_recon_const_outres_j = np.tile(batch_recon_const_outres[j, ...].numpy(), (batchsize, 1, 1, 1))

        input_color = torch.from_numpy(batch_j).cuda()
        input_greylevel = torch.from_numpy(batch_recon_const_j).cuda()

        # Get mean from GMM
        curr_mu = gmm_mu[j * nmix : (j + 1) * nmix, :]
        orderid = np.argsort(gmm_pi[j * nmix : (j + 1) * nmix, 0].cpu().data.numpy().reshape(-1))

        # Sample from GMM
        z = curr_mu.repeat(int((batchsize * 1.0) / nmix), 1)

        # Predict color
        _, _, color_out = model_vae(input_color, input_greylevel, z)

        data.saveoutput_gt(
            color_out.cpu().data.numpy()[orderid, ...],
            batch_j[orderid, ...],
            "divcolor_%05d_%05d" % (batch_idx, j),
            nmix,
            net_recon_const=batch_recon_const_outres_j[orderid, ...],
        )

print("Complete inference")
```

Outline

- VAE Revision
- Image Colorization Overview
- VAE-based Image Colorization Model
- Implementation

Summary

- ✓ Studied Variantional Autoencoder
- ✓ Studied Image Colorization Overview
- ✓ Studied VAE-based Image Colorization