CPEN 455 Final Project

Riley Zhang

Abstract

I created a conditional PixelCNN model using the PyTorch Embedding module to represent label information into a high dimensional vector space, which is then fused with the pixel data after applying the initial convolutional layer in the model. Using this strategy, the model managed to achieve a 77% classification accuracy on the given test set.

1 Model

The first main change to the model is the implementation of label embedding. The embedding layer is initialized using the nn.Embedding module, where the size of the embedding dictionary is equal to the number of classes, and we make the embedding dimension to $C_{init} \times H \times W$. This is then resized to fit the dimensions of the pixel information. C_{init} is defined to be the channel size after the initial convolution, the layers which are defined as u_init and ul_init.

It was found that fusing our encoding after the pixels passed through the initial convolutional layer had the best effect, most likely because of the larger channel size which was increased from our input channel size of 3 to 20 and allowed our labels to affect a larger feature map. Figure 1 illustrates where our encoding is fused in the model. The code for embedding and fusion are shown in Figure 2.

The second change to the codebase was adding a classifier function. This classifier function generates an output x_out for each class, then the modified loss function $discretized\_mix\_logistic\_loss$ (modified to calculate loss over one image instead of over a batch) calculates the loss between our input and output. For each image, the label corresponding with the lowest loss is predicted. The code for the classifier function is shown in Figure 3.

2 Experiments

Most of the work was experimenting with different encoding methods and where in the model to fuse label information. The validation accuracy for many of these different experiments is shown in Figure 4.

2.1 Encoding Methods

Two encoding methods were initially created and tested. Absolute Positional Encoding (APE) derived from PA2, and using the nn.Embedding module which was eventually picked.

APE was implemented by first doing a one-hot encoding of the label tuple, which was then multiplied by a encoding weight matrix to match the pixel dimensions. This result was then passed to the APE function and added to x. It was shown to immediately perform worse than nn.Embedding in the same location, with a validation accuracy of about 5% lower than nn.Embedding, and thus was abandoned.

2.2 Fuse Location

Deciding on fuse locations was done by simply comparing validation accuracy between implementations. There were many different fuse locations possible within the model, listed with their label in Figure 4:

- Fuse with x before any convolutions (nn fuse x)
- Fuse with x after initial convolution (nn fuse after init)
- Fuse with x_out at the end of the model (nn fuse_out)
- Fuse right before the down pass (nn fuse between)
- Fuse during the down pass (nn fuse at down)

As shown from the validation accuracy, fusion after the initial convolution and simply fusing with x before any convolutions were the most competitive. However, after running these two versions, the former began overtaking after around 50 epochs, and thus it was chosen.

3 Conclusion

After researching to complete this project, I gained a lot of insight onto conditional models as a whole, and definitely has changed from when I started this project, when I did not know that a conditional model could be created from an unconditional one simply by adding label information.

For a future study, I would like to investigate how to implement the encoding solution implemented in the paper "Conditional Image Generation with PixelCNN Decoders" and am curious on this specific model's performance compared to mine.

4 Appendix

```python
def forward(self, x, labels, sample=False):
    # find dimensions
    B, D, H, W = x.shape
    device = x.device

    indices = []

    # change all labels into indices
    for label in labels:
        if label == "Class0":
            indices.append(0)
        elif label == "Class1":
            indices.append(1)
        elif label == "Class2":
            indices.append(2)
        else:
            indices.append(3)

    label_embed = torch.LongTensor(indices).to(device)

    label_embed = self.embedding(label_embed).to(device)

    # reshape to match pixel info
    label_embed = label_embed.reshape(B, self.nr_filters, H, W)

    if self.init_padding is not sample:
        xs = [int(y) for y in x.size()]
        padding = Variable(torch.ones(xs[0], 1, xs[2], xs[3]), requires_grad=F
        self.init_padding = padding.cuda() if x.is_cuda else padding

    if sample :
        xs = [int(y) for y in x.size()]
        padding = Variable(torch.ones(xs[0], 1, xs[2], xs[3]), requires_grad=F
        padding = padding.cuda() if x.is_cuda else padding
        x = torch.cat((x, padding), 1)

    ###      UP PASS    ###
    x = x if sample else torch.cat((x, self.init_padding), 1)
    # fuse label information with pixels
    u_list  = [self.u_init(x) + label_embed]
    ul_list = [self.ul_init[0](x) + self.ul_init[1](x) + label_embed]

    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])]
```

Figure 1: Label embedding and fusion code.

```python
def classify(self, x, num_classes):
    B, D, H, W = x.shape
    device = x.device
    my_bidict = {'Class0': 0,
                 'Class1': 1,
                 'Class2': 2,
                 'Class3': 3}

    # And get the predicted label, which is a tensor of shape (batch_size,)
    # initialize predictions and losses
    y_pred = torch.zeros((B, ))
    y_losses = torch.full((B, ), float('inf'))

    for key in my_bidict.keys():
        label = (key, ) * B
        model_output = self(x, label)

        losses = (discretized_mix_logistic_loss(x, model_output, batch=False))

        # if the loss is lower, replace in the tensor
        for i in range(0, B):
            if losses[i] < y_losses[i]:
                y_losses[i] = losses[i]
                y_pred[i] = my_bidict[key]

    return y_pred, y_losses
```

Figure 2: Classifier code.

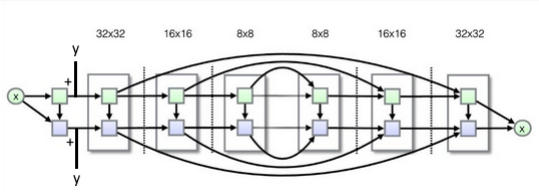

Figure 3: Illustration of fusion location (original sourced from TAs).

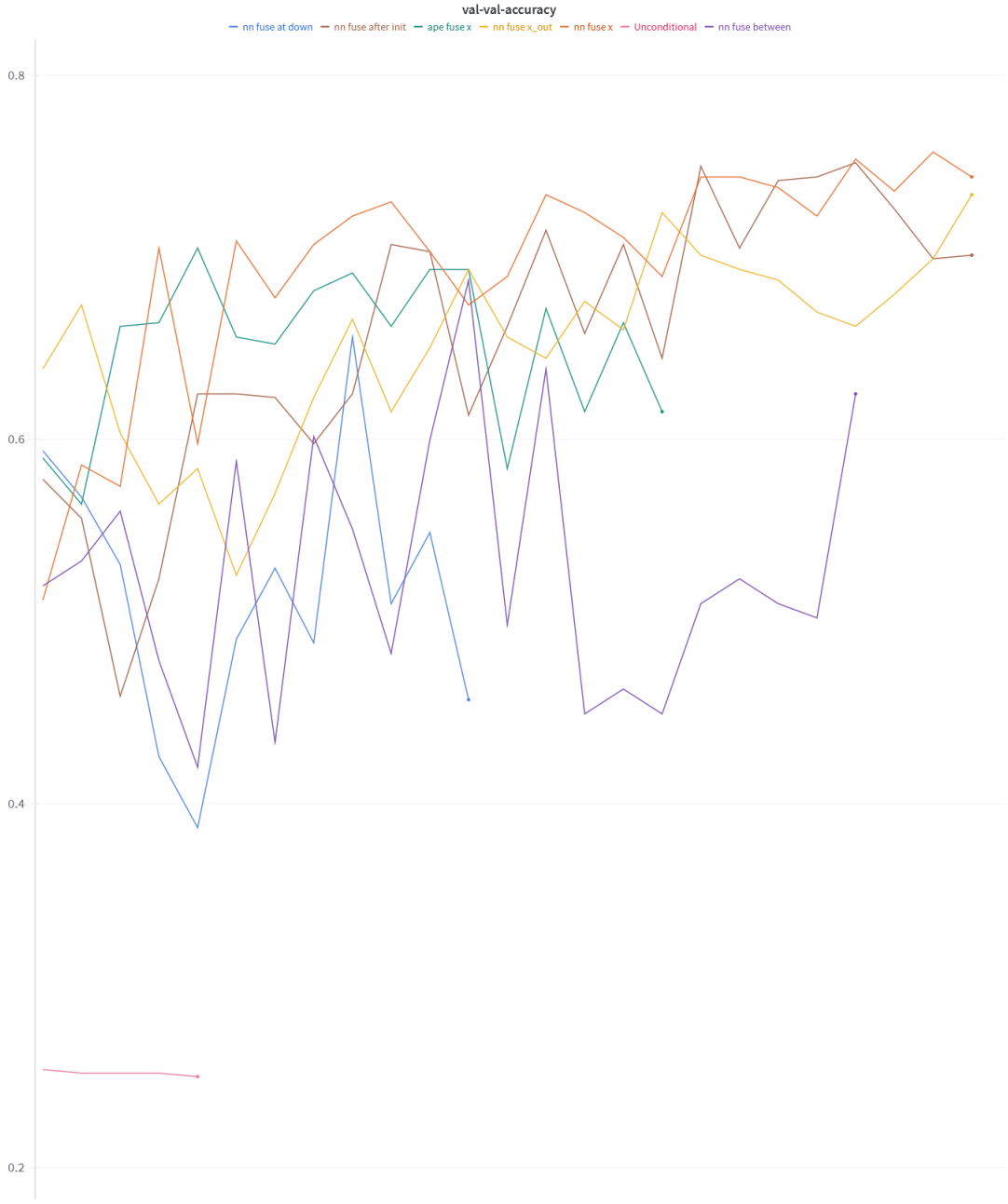

Figure 4: Validation accuracy of different fusion locations and encoding strategies.