NTNU

Kunnskap for en bedre verden

Department of Electronic Systems

TTT4135 - Multimedia Signal Processing

Assignment 3

Authors:
Vegard Iversen, Sondre Olimb, Sebastian Skogen Raa

Feb, 2022

Table of Contents

Code

All code can be seen on github here: https://github.com/Eeaeau/TTT4135-Multimedia-Signal-Processing/blob/main/Assignment%203

1 Problem 1 Information theory

1a

From the lectures we have been given that for a finite, discrete random variable X having a set of outcomes $\{x_i\}$ with probabilities $p_i = P_X(X = x_i)$, we have

$$H(X) = -\sum_i p_i \log_2 p_i = E\left[-\log_2 p_i\right] \tag{1}$$

where $E[]$ is the expectation operator.

The entropy for the first distribution then becomes: $P(X = x) = 0.5$ for $x \in \{0, 1\}$

$$H(X) = -P(X = 0)\log_2 P(X = 0) - P(X = 1)\log_2 P(X = 1) = -2(0.5\log_2 0.5) = 1 \tag{2}$$

The second distribution:

$P(X = 0) = 0, P(X = x)$ for $x \in \{1, 2\}$

$$H(x) - P(X = 0)\log_2 P(X = 0) - P(X = 1)\log_2 P(X = 1) - P(X = 2)\log_2 P(X = 2)$$

As shown below, the first term disappears and we have tha same entropy as the previous distibution.

$$H(x) = -0\log_2 0 - 0.5\log_2 0.5 - 0.5\log_2 0.5 = 1 \tag{3}$$

The third distibution:

$$P(X = 0, Y = 0) = 1/2, P(X = 0, Y = 1) = 1/4, P(X = 1, Y = 0) = 1/8, P(X = 1, Y = 1) = 1/8 \tag{4}$$

$$H(X, Y) = -\sum(P(X, Y)log_2 P(X, Y)$$

Since it is tedious and trivial, we will skip the interim calculations.

$$H(X, Y) = -(0.5log_2(0, 5) + 0.25log_2(0.25) + 0.125log_2(0.125) + 0.125log_2(0.125)) = 1.75 \tag{5}$$

The final distribution:

$P(X = n) = 2^{-n}, n \in \{1, 2, 3, ..., \infty\}$

$$\begin{aligned} H(X) &= -\sum_{n=1}^{\infty} P(X = n)log_2 P(X = n) = -\sum_{n=1}^{\infty} 2^{-n} log_2 2^{-n} \\ &\Rightarrow -\sum_{n=1}^{\infty} -2^{-n} n log_2 2 = \sum_{n=1}^{\infty} 2^{-n} n = 2 \end{aligned} \tag{6}$$

The solution of the last equation was derived using wolfram.

1b

H(x):

$H(x) = -P(X=0)log_2P(X=0) - P(X=1)log_2P(X=1)$

$P(X=0) = P(X=0,Y=0) + P(X=0,Y=1)$

$P(X=1) = 1 - P(0.5)$

The joint probability:

$P(X=0,Y=0) = P(X=0|Y=0)P(Y=0) = 1/8$

$P(X=0,Y=1) = P(X=0|Y=1)P(Y=1) = 3/8$

where P(Y=1) = 1-P(Y=0) =3/4 and

P(X=0—Y=1)= 1 - P(X=1—Y=1) = 0.5

solving P(X=0) and P(X=1):

$P(X=0) = P(X=0,Y=0) + P(X=0,Y=1) = 1/2$

$P(X=1) = 1 - P(x=0) = 1/2$

H(x) then becomes

$$H(x) = -P(X=0)log_2P(X=0) - P(X=1)log_2P(X=1) = -log(0.5) = 1 \tag{7}$$

H(Y):

$$H(Y) = -P(y=0)log_2P(y=0) - P(y=1)log_2P(y=1) = -(0.25log_2(0.25) + 0.75log_2(0.75) = 0.811 \tag{8}$$

H(X,Y)

$$H(X,Y) = -\sum P(X,Y)log_2P(X,Y) \tag{9}$$

from earlier we know:

$P(X=0,Y=0) = 1/8$

$P(X=0,Y=1) = 3/8$

So we need the folowing distibutions:

$P(X=1,Y=0) = P(X=1|Y=0)P(Y=0) = 1/8$

where P(X=1—Y=0) = 1- P(X=0—Y=0) =0.5

$P(X=1,Y=1) = P(X=1|Y=1)P(Y=1) = 3/8$

$-P(X=0,Y=0)log_2P(X=0,Y=0) = 0.375$

$-P(X=0,Y=1)log_2P(X=0,Y=1) = 0.531$

$-P(X=1,Y=0)log_2P(X=1,Y=0) = 0.375$

$-P(X=1,Y=1)log_2P(X=1,Y=1) = 0.531$

$$H(X,Y) = -\sum P(X,Y)log_2P(X,Y) = 1.812 \tag{10}$$

H(X|Y)

The Conditional Entropy

$$H(X|Y) = -\sum_m \sum_n P(Y,X) \log_2 P(X|Y) \tag{11}$$

we have:

$P(X = 0|Y = 0) = 0.5$

$P(X = 0|Y = 1) = 0.5$

$P(X = 1|Y = 0) = 0.5$

$P(X = 1|Y = 1) = 0.5$

Then entropy then becomes

$H(X = 0|Y = 0) = -P(Y = 0, X = 0) \log_2 P(X = 0|Y = 0) = 0.25$

$H(X = 0|Y = 1) = -P(Y = 0, X = 1) \log_2 P(X = 1|Y = 0) = 0.25$

$H(X = 1|Y = 0) = -P(Y = 1, X = 0) \log_2 P(X = 0|Y = 1) = 0.25$

$H(X = 0|Y = 0) = -P(Y = 1, X = 1) \log_2 P(X = 1|Y = 1) = 0.25$

$$H(X|Y) = -\sum P(X|Y) log_2 P(X|Y) = 1 \tag{12}$$

H(Y|X)

$$H(Y|X) = -\sum P(X|Y) log_2 P(Y|X) \tag{13}$$

Need to find the distibutions:

$P(Y = 0|X = 0) = P(X = 0|Y = 0)P(Y = 0)/P(X = 0) = 0.25$

$P(Y = 0|X = 1) = P(X = 1|Y = 0)P(Y = 0)/P(X = 1) = 0.25$

$P(Y = 1|X = 0) = P(X = 0|Y = 1)P(Y = 1)/(P(X = 0) = 0.75$

$P(Y = 1|X = 1) = P(X = 1|Y = 1)P(Y = 1)/P(X = 1) = 0.75$

The Entopy becomes:

$H(x = 0|Y = 0) = -P(X = 0, Y = 0) \log_2 P(Y = 0|X = 0) = 0.25$

$H(X = 0|Y = 1) = -P(X = 0, Y = 1) \log_2 P(Y = 1|X = 0) = 0.156$

$H(X = 1|Y = 0) = -P(X = 1, Y = 0) \log_2 P(Y = 0|X = 1) = 0.25$

$H(X = 0|Y = 0) = -P(X = 1, Y = 1) \log_2 P(Y = 1|X = 1) = 0.156$

$$H(Y|X) = -\sum P(Y|X) log_2 P(Y|X) = 0.812 \tag{14}$$

I(X;Y)

From the lecture notes we find $I(X;Y) = H(X) - H(X|Y)$ We then get:

$$I(X;Y) = H(X) - H(X|Y) = 1 - 1 = 0 \tag{15}$$

Summary

H(X)	1
H(Y)	0.811
H(X,Y)	1.812
H(X | Y)	1
H(Y|X)	0.812
I(X;Y)	0

1C

We can calculate the stady state of the marklov chain from $\pi = \pi P$ where π is the stady state distibution vector $[\pi_0, \pi_1]$ and P is the tranition matrix.**markov**

Firsly lets find the transition matrix, given in equation (16)

$$P = \begin{bmatrix} P(X_n = 0|X_{n-1} = 0) & P(X_n = 1|X_{n-1} = 0) \\ P(X_n = 0|X_{n-1} = 1) & P(X_n = 1|X_{n-1} = 1) \end{bmatrix} = \begin{bmatrix} 7/8 & ? \\ ? & 7/8 \end{bmatrix}$$

(16)

Tow of the four unknown is given in the assignment, so we need to find the remaining two:

$P(X_n = 1|X_{n-1} = 0) = 1 - P(X_n = 0|X_{n-1} = 0) = 1/8$

$P(X_n = 0|X_{n-1} = 1) = 1 - P(X_n = 1|X_{n-1} = 1) = 1/8$

We then get the final transition matrix which we can then find the steady state.

$$P = \begin{bmatrix} P(X_n = 0|X_{n-1} = 0) & P(X_n = 1|X_{n-1} = 0) \\ P(X_n = 0|X_{n-1} = 1) & P(X_n = 1|X_{n-1} = 1) \end{bmatrix} = \begin{bmatrix} 7/8 & 1/8 \\ 1/8 & 7/8 \end{bmatrix} \tag{17}$$

We can now compute the relation between the steady state distribution π and transition matrix P. $\pi = \pi P$

This gives us the following two equations:

$$\begin{aligned} \frac{7}{8}\pi_0 + \frac{1}{8}\pi_1 &= \pi_0 \\ \frac{1}{8}\pi_0 + \frac{7}{8}\pi_1 &= \pi_1 \end{aligned} \tag{18}$$

We have two equations with two unknowns, solving with subtraction.

$6/8\pi_0 - 6/8\pi_1 = \pi_0 - \pi_1 \rightarrow \pi_0 = \pi_1$

By using this result and the fact that $\sum \pi_i = 1$ we find that the steady steate disruption is given by**markov**:

$$\pi = [1/2, 1/2]$$

Finally we then need to find the entropy rate. From the lecture notes we have:

If iid. random variables X_i we have

$$H(\mathcal{X}) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n) \tag{19}$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} H(X_i) \tag{20}$$

$$= H(X_1) \tag{21}$$

I.e we need to find $H(x_1)$

The entopy rate for a markovchain then becomes **entropy**:

$$H(X_1) = -\sum_{i,j} \pi_i P_{i,j} log(P_{i,j})$$

$-0.5(\frac{1}{8} log_2(\frac{1}{8})) = 0.1875$

$-0.5(\frac{7}{8} log_2(\frac{7}{8}) = 0.084$

$$H(X_1) = 2(0.1875 + 0.084) = 0.543$$

1g

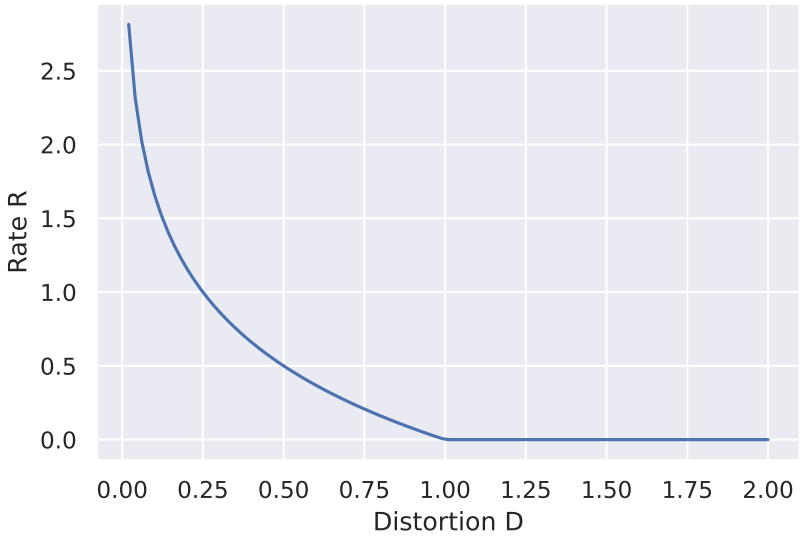

Figure 1: Plot of RD curve for the information source.

1e

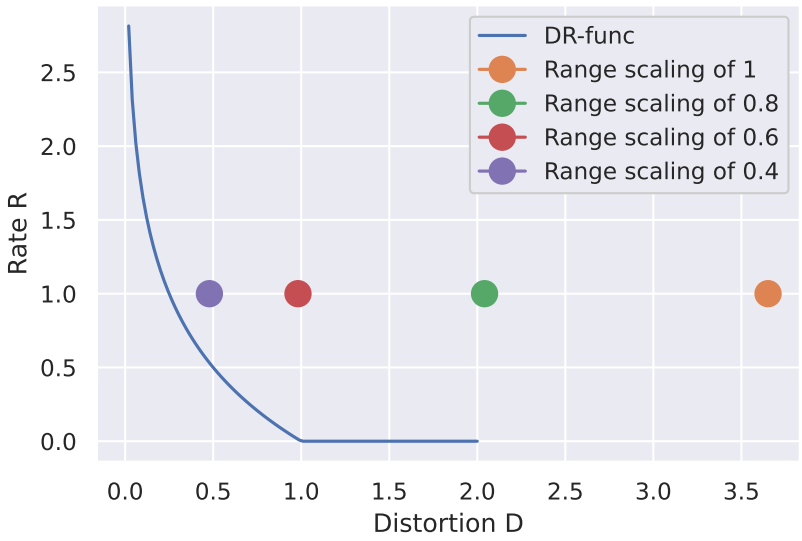

Figure 2: Plot of RD pairs alongside RD curve with different quantization ranges of max and min value at 1 bit.

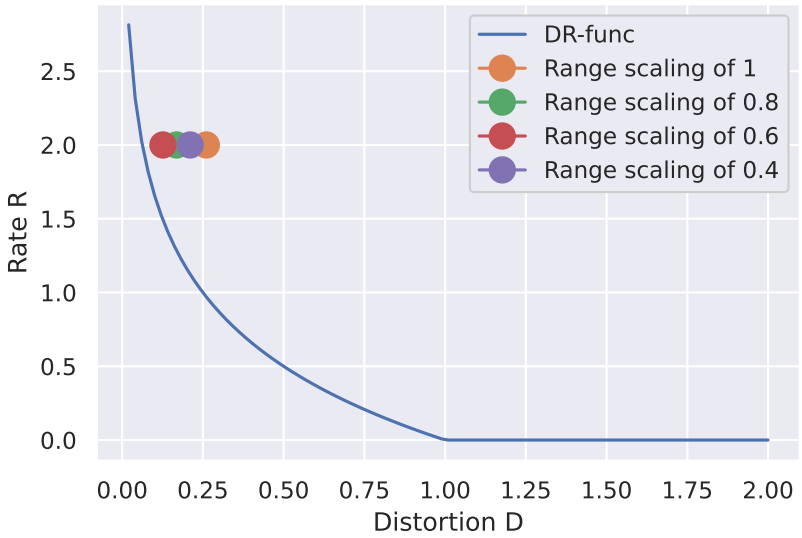

Figure 3: Plot of RD pairs alongside RD curve with different quantization ranges of max and min value at 2 bit.

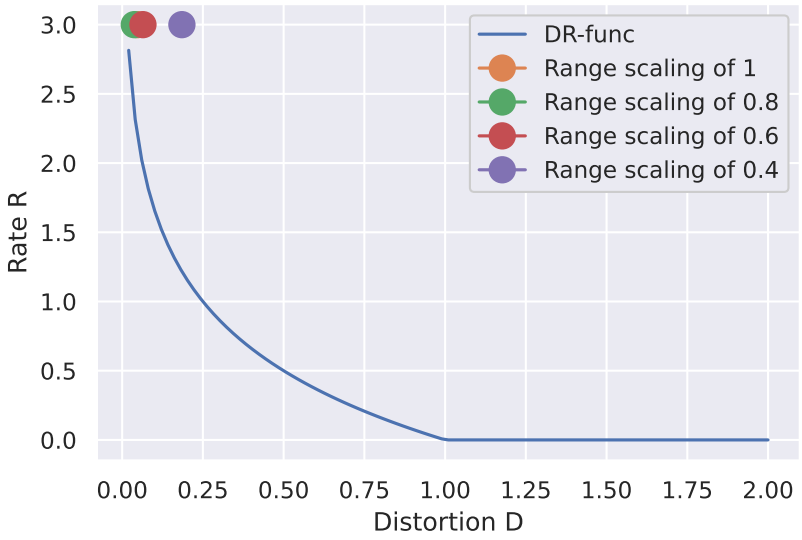

Figure 4: Plot of RD pairs alongside RD curve with different quantization ranges of max and min value at 3 bit.

1f

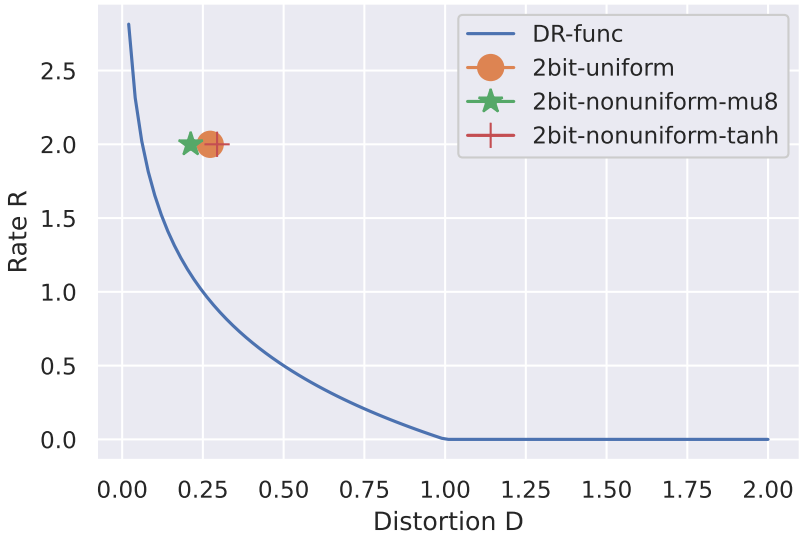

Figure 5: Plot of RD pairs alongside RD curve with different encoding functions at 2 bit.

1.1 1g

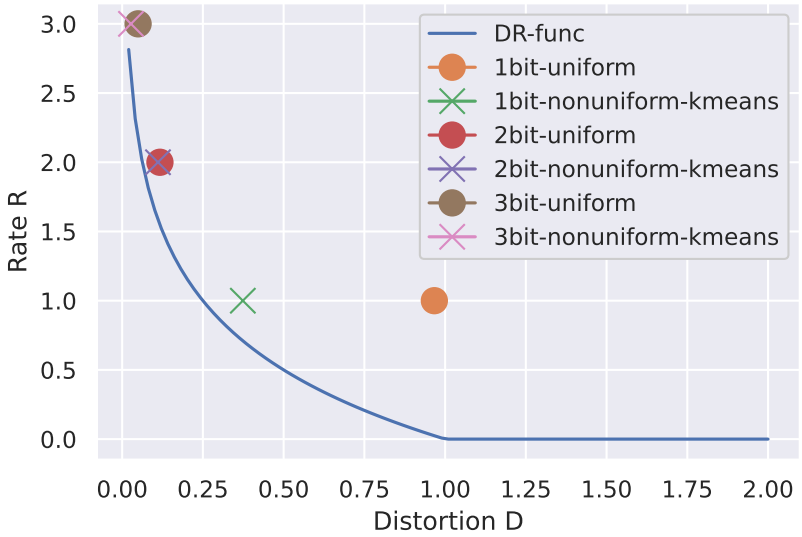

Figure 6: RD pair of kmean compared to linear quantizer plotted alongside RD curve.

2 Problem 2 Deep neural networks

In order to achieve acceptable training times we used Google Colab in order to access GPU power. We also changed the code to use the tempalte provided in TDT4265 - Computer Vision and Deep Learning **git˙start**. Part of the code is witten by us but large parts is a template provided in the course. However the neural network, and all other improvements where developed by us.

1. The template script achived a test accuracy of about 60% and took about 8 minutes

2. Randomly splitting the the training set in to validation and training. SubsetRandomSampler is a PyTotch package

```
indices = list(range(len(data_train)))
    split_idx = int(np.floor(validation_fraction * len(data_train)))

    val_indices = np.random.choice(indices, size=split_idx, replace=False)
    train_indices = list(set(indices) - set(val_indices))

    train_sampler = SubsetRandomSampler(train_indices)
    validation_sampler = SubsetRandomSampler(val_indices)

    dataloader_train = torch.utils.data.DataLoader(data_train,
                                                   sampler=train_sampler,
                                                   batch_size=batch_size,
                                                   num_workers=2,
                                                   drop_last=True)

    dataloader_val = torch.utils.data.DataLoader(data_train,
                                                 sampler=validation_sampler,
                                                 batch_size=batch_size,
                                                 num_workers=2)
```

2.1 Improving the neural network

In order to improve the neural network we utilized what we have learned in the course. The changes are quite extensive so we will go trough them one by one.

1. Changed the neural network

Firstly we drastically changed the neural network.

In the feature extractor we followed a general architecture, conv, batch-norm, relu, conv, relu, maxPool,dropout. This pattern repeats itself for each "layer" with ever increasing size. The classifier is mostly the same with liner, relu and dropout.

```
self.feature_extractor = nn.Sequential(
        #Layer 1
        nn.Conv2d(in_channels=3, out_channels=num_filters, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_filters),
        nn.ReLU(),


        #Layer 2
        nn.Conv2d(in_channels=num_filters, out_channels=num_filters*2, kernel_size=3, padding=1),
        nn.ReLU(),

        nn.MaxPool2d(kernel_size=[2,2], stride=2),
        nn.Dropout2d(p=0.1),

        # Layer 3
        nn.Conv2d(in_channels=num_filters*2, out_channels=num_filters*3, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_filters*3),
        nn.ReLU(),


        #Layer 4
        nn.Conv2d(in_channels=num_filters*3, out_channels=num_filters*4, kernel_size=3, padding=1),
        nn.ReLU(),

        nn.MaxPool2d(kernel_size=[2,2], stride=2),
        nn.Dropout2d(p=0.1),

        # Layer 5
        nn.Conv2d(in_channels=num_filters*4, out_channels=num_filters*5, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_filters*5),
        nn.ReLU(),


        #Layer 7
        nn.Conv2d(in_channels=num_filters*5, out_channels=num_filters*6, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=[2,2], stride=2),
        nn.Dropout2d(p=0.1),

        #layer 8
        nn.Conv2d(in_channels=num_filters*6, out_channels=num_filters*7, kernel_size=3, padding=1),
        nn.BatchNorm2d(num_filters*7),
        nn.ReLU(),

        #Layer 9
        nn.Conv2d(in_channels=num_filters*7, out_channels=num_filters*8, kernel_size=3, padding=1),
        #nn.ELU(),
        nn.MaxPool2d(kernel_size=[2,2], stride=2),
        nn.Dropout2d(p=0.1)
        )


        self.classifier = nn.Sequential(


            nn.Linear(self.num_output_features, 512),
            nn.ReLU(),
```

```
            nn.Dropout(p=0.2),
            nn.Linear(512, 256),
            nn.ReLU(),


            nn.Dropout(p=0.2),
            nn.Linear(256, 10)
        )
```

After training for 30 epochs, with batch size 64 we achieved a test accuracy of about 60%. So basically no improvement. This is no surprise since the network has grown to over 2 million parameters, therefor SGD will be quite slow to converge. This we can see in the validation loss and accuracy plot blow where the plot dosent converge. So the next logical step will be to change the optimizer.

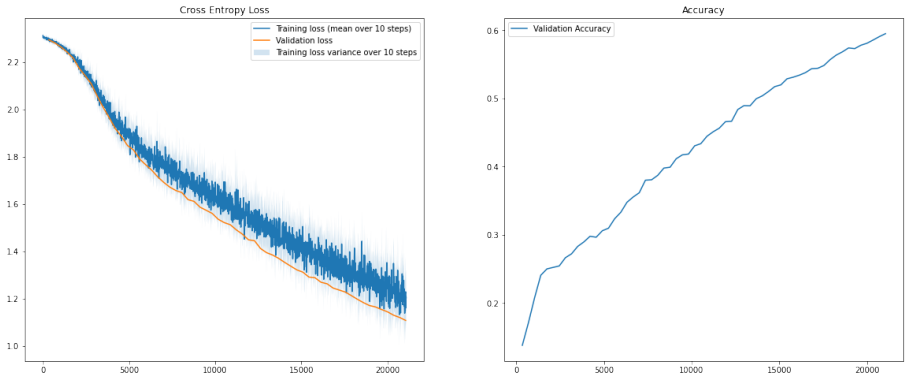

Figure 7: Plot of validation loss and accuracy for the new NN

2. Change to the ADAM optimizer

In order the archive faster convergence we change to the Adam optimizer. This increased the test accuracy to 83%. We can also we in the plot below we have a much faster convergence of the loss. in converges and Early stops at 7000 iterations, compared to no covergance and early stopping at 20000 iterations with the SGD optimizer.

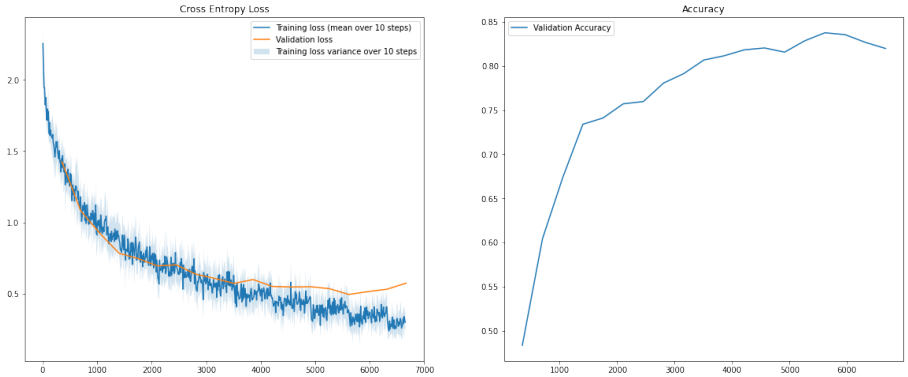

Figure 8: Plot of validation loss and accuracy for the new NN with ADAM

3. Dynamic learning rate

In order to improve it further we add data a dynamic learning rate. If the early stopping criteria is met once we change the learning rate with torch.optim.lr_scheduler. ExponentialLR(self.optimizer, gamma=0.9, verbose=True). If the early stopping criteria is met twice in a row we stop training. This means if we change the learning rate and see an improvement in the loss, we continue training,

otherwise we stop. With this we further improved the test accuracy to 85%. However we can clearly sign of overfitting in the plot below as the test and validation loss is diverging.

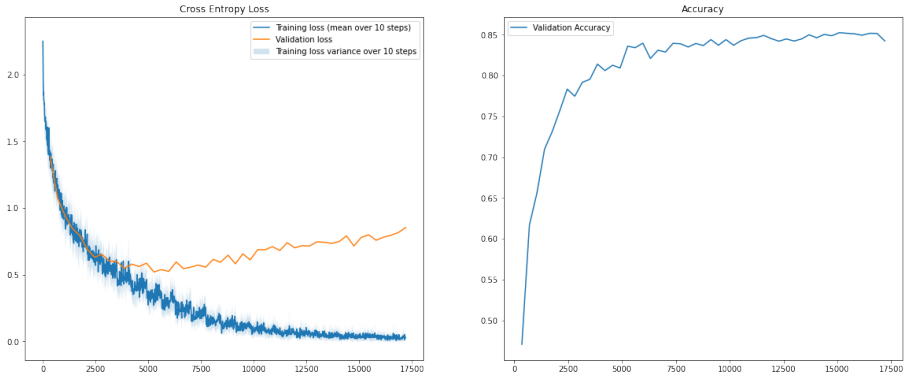

Figure 9: Plot of validation loss and accuracy for the new NN with Dynamic learning rate

4. Data augmentation

In order to try and prevent the NN from over training we add data augemntation. PyTorch as a built in data augmentation for the CIGFAR10 dataset, so we decided to use this. Why should this work in theory? Well by using data augmentation we artificially increase the size of the trainingset, meaning that it will be harder for the NN to fit to local variation in the data set, and we should get a better fit to global trends. As we can see in the plot below, we managed to reduce the overfitting, quite significantly. We also increased the test accuracy to 86%

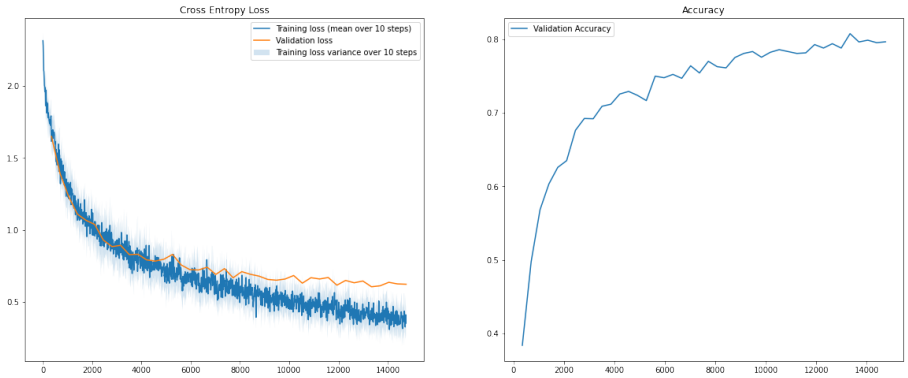

Figure 10: Plot of validation loss and accuracy for the new NN with Data aumentation

2.2 Final result

Validation Loss: 0.59 Test accuracy 86% We have increased the test accuracy from 60% to 86% which we find to be satisfactory within the confines of this task.

2.3 PR curve

In order to calculate the PR curve we used the scikit learn function sklearn.metrics.precision_recall_curve

```
1
2 precision = dict()
3 recall = dict()
4 for i in range(10):
5
6   precision[i], recall[i], _ = precision_recall_curve(y_test[:,i], y_score[:, i])
```

```
7
8  classes = ('plane', 'car', 'bird', 'cat', 'deer',
9             'dog', 'frog', 'horse', 'ship', 'truck')
10 fig, ax = plt.subplots()
11 for i in range(10):
12   ax.plot(recall[i], precision[i],label = classes[i])
13   print(f'class:{classes[i]}',auc(recall[i], precision[i]))
14
15
16 #add axis labels to plot
17 ax.set_title('Precision-Recall Curve')
18 ax.set_ylabel('Precision')
19 ax.set_xlabel('Recall')
20 ax.legend()
21
22 #display plot
```

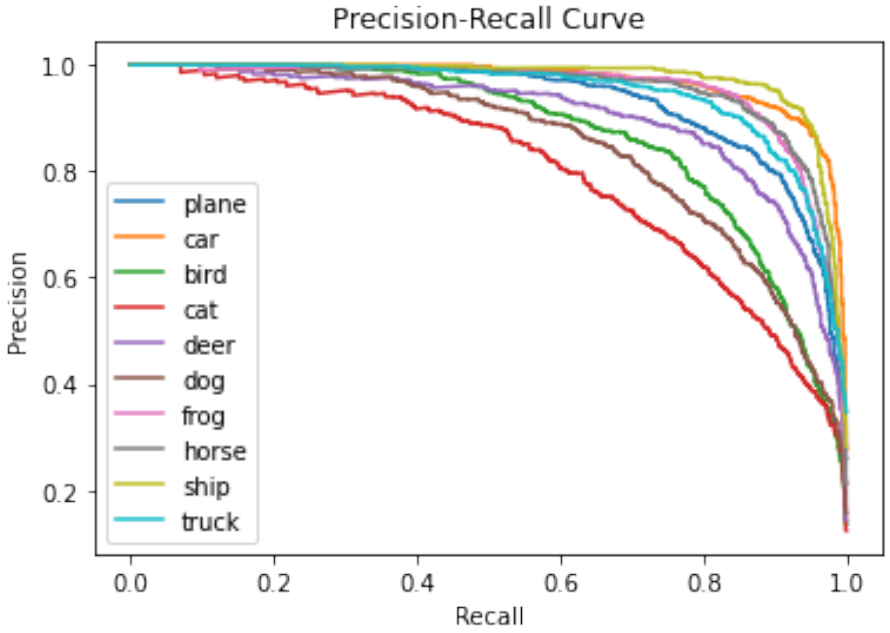

Figure 11: Plot of pr curve for the new model

To calculate area under the curve we used the scikit function sklearn.metrics.auc.

Table 1: Area Under the Curve

Class	**AUC**
plane	0.93
car	0.96
bird	0.84
cat	0.74
deer	0.89
dog	0.82
frog	0.95
horse	0.95
ship	0.97
truck	0.94

2.4 Testing a ResNet approach

We also tried to create a ResNet (with skip layers) to train on the dataset. This gave very bad result. One of the reasons for this is the small dataset compared to imagenet, which the original ResNet are trained on. The model can be seen on github on vegpro branch (if not merged with main)