BBM418: Computer Vision Lab. **(Due: 12/06/2020)**

Assignment 4

Instructor: Assoc. Prof. Dr. Nazlı İkizler Cinbiş, Research Assistan Burçak Asal *Name:* Deniz Zağlı, *Netid:* 21527668

- **Introduction:**

  In this assignment, we are asked to classify the American Sign Language gestures in the videos. There is a dataset from youtube videos for this task. I use this dataset in this project. There is also a JSON file that shows which movements are between the seconds in the videos. What makes this classification problem difficult is that a single frame has no meaning. In other words, if we do not know before and after a movement, we find it difficult to predict the movement. In order to solve this problem, I use the flow data I produced using frames. In this way, I have an idea about before and after the Movement and make the classification. In this assignment, I will train 4 different NETs and compare the results. These are: Spatial ConvNET, Temporal ConvNET, Early Fusion, Late Fusion.

- **Implementation Details:**

  Pre-Processing:

  When creating a dataset, I used JSON file as a priority. While reading the data in the JSON file, I paid attention to two factors. First, I received data whose label value is less than 9 or 9. Second, I got the data with the url of the videos in the video dataset. I applied these two processes for train, validation and testing. As a result, I kept the url, start time, end time, label information for each data. Then, using this information, I first created the frames. He created a new dataset for this. I added the google drive link of the frame dataset I created to the below. I used the VideoCapture function of the cv2 library when creating the frames. I used the frame given to obtain the optical flow data required for the Temporal Convolutional Neural Network. I also created a separate dataset for optical flow data. I added the google drive link of this dataset I created to the below. I used the calcOpticalFlowFarneback function of the cv2 library when creating the optical flow data. At the end of these stages, my data is ready for 4 different neural networks that I want to create.

  https://drive.google.com/drive/folders/17dmk2S_RR9ZZAfPo5tlYAaCtJVtFbAbF?usp=sharing

  https://drive.google.com/drive/folders/1qk8EasDKwIE-lGsVZiUfToEj3Cb9zwJz?usp=sharing

  Spatial ConvNET:

  I used dataset with frames for Spatial ConvNET. There was a problem like this. There is more than one frame for each input file, but Spatial ConvNET uses only one frame. For this reason, I had to select only one of the frames in the input. I divided the number of frames in that input in half and chose the middle frame. For this reason, I provided the required input frame for Spatial ConvNET. I read this input frame, which I selected from the middle of the frames, in RGB format and resized it as 224x224. I kept it in numpy array to be called in this resized Dataset class. The model I created for Spatial ConvNET is shown in figure 1.

  Temporal ConvNET

  I used flows for Temporal ConvNET. In other words, 18 images were required as input. I used the flows in the Optical Flows dataset I created. This dataset contains 18 inputs required for each input in each file. I read these images as greyscale and then resized them as 224x224. I made the resized images ready for Dataset Class. The model I created for Tempral ConvNET is shown in figrure 2. In fact, according to Spatial ConvNET, the only difference is that 18 channels are selected instead of 3.

  Early Fusion

  During the dataset reading stage in the early Fusion section, I did the same file reading operations as I explained in the Spatial ConvNET and Temporal ConvNET sections. Because for this stage, we need both frame and flow inputs. I updated my Dataset Class so that it can read two different inputs. In the model part, the training is actually the same as spatial ConvNET and Temporal Convnet until the last fully connected layer stage. Within the same model, these parts are trained separately. Before the last fully connected layera is connected, the fusion is done and they give a single output. The model I created

```
[ ]  class CNN(nn.Module):
        def __init__(self):
            super(CNN, self).__init__()
            self.layer1 = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.layer2 = nn.Sequential(
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(200704, 1000)
            self.fc2 = nn.Linear(1000, 10)

        def forward(self, x):
            out = self.layer1(x)
            out = self.layer2(out)
            out = out.reshape(out.size(0), -1)
            out = self.drop_out(out)
            out = self.fc1(out)
            out = self.fc2(out)
            return out
```

Figure 1: Spatial ConvNET Model

```
[ ]  class CNN(nn.Module):
        def __init__(self):
            super(CNN, self).__init__()
            self.layer1 = nn.Sequential(
                nn.Conv2d(18, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.layer2 = nn.Sequential(
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(200704, 1000)
            self.fc2 = nn.Linear(1000, 10)

        def forward(self, x):
            out = self.layer1(x)
            out = self.layer2(out)
            out = out.reshape(out.size(0), -1)
            out = self.drop_out(out)
            out = self.fc1(out)
            out = self.fc2(out)
            return out
```

Figure 2: Temporal ConvNET Model

for early fusion is shown in figure 3. I would like to mention that I used a single model, a single criterion, a single optimizer for Early Fusion.

Late Fusion

The process of reading data in the Late Fusion phase is the same as in Early fusion. The Dataset Glass I created is the same as Early Fusion. But I created two separate models in Late Fusion. Thanks to two different models, the lostlar softmax result is collected as a result of spatial and temporal convnet. And as a result of this total, the maximum value is calculated. For two separate models, I calculated a single loss value. And backpropagation takes place with this joint loss value. I used two separate criterions and two optimizers for the two models. The model I used for Late Fusiopn is shown in figure 4.

- **Experiments** While performing my experiments, I tested two parameters. Bach Size and Learning Rate. While performing these operations, I kept the number of epoch at 10. I selected the best model by testing the experiments separately for each model and interpreting the results I obtained. In the first stage, I tested the batch size for 4 different models. While doing this, I determined the number of epoch as 10 and the learning rate as 0.00001.

  Spatial ConvNET - Batch Size Experiments:

  1) Batch Size 4: Test Accuracy: % 16.30 Validation Accuracy: % 12.82 (Figure 5, Figure 6)

  2) Batch Size 8: Test Accuracy: % 11.46 Validation Accuracy: % 12.68 (Figure 7, Figure 8)

  3) Batch Size 16: Test Accuracy: % 19.10 Validation Accuracy: % 14.74 (Figure 9, Figure 10)

```
[ ]  class CNN(nn.Module):
        def __init__(self):
            super(CNN, self).__init__()
            self.layer1_spatial = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.layer1_flow = nn.Sequential(
                nn.Conv2d(18, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(401408, 1000)
            self.fc2 = nn.Linear(1000, 10)

        def forward(self, x, y):
            x1 = self.layer1_spatial(x)
            x1 = x1.view(x1.size(0), -1)
            x2 = self.layer1_flow(y)
            x2 = x2.view(x2.size(0), -1)
            fusion = torch.cat((x1, x2), dim=1)
            out = fusion.reshape(fusion.size(0), -1)
            out = self.drop_out(out)
            out = self.fc1(out)
            out = self.fc2(out)
            return out
```

Figure 3: Early Fusion Model

4) Batch Size 32: Test Accuracy: % 15.33 Validation Accuracy: % 13.60 (Figure 11, Figure 12)

Temporal ConvNET - Batch Size Experiments:

1) Batch Size 4: Test Accuracy: % 23.91 Validation Accuracy: % 19.10 (Figure 13, Figure 14)

2) Batch Size 8: Test Accuracy: % 22.92 Validation Accuracy: % 20.47 (Figure 15, Figure 16)

3) Batch Size 16: Test Accuracy: % 24.64 Validation Accuracy: % 23.52 (Figure 17, Figure 18)

4) Batch Size 32: Test Accuracy: % 19.49 Validation Accuracy: % 21.22 (Figure 19, Figure 20)

Early Fusion - Batch Size Experiments:

1) Batch Size 4: Test Accuracy: % 35.87 Validation Accuracy: % 23.64 (Figure 21, Figure 22)

2) Batch Size 8: Test Accuracy: % 27.08 Validation Accuracy: % 20.71 (Figure 23, Figure 24)

3) Batch Size 16: Test Accuracy: % 22.56 Validation Accuracy: % 19.15 (Figure 25, Figure 26)

4) Batch Size 32: Test Accuracy: % 19.49 Validation Accuracy: % 16.73 (Figure 27, Figure 28)

Late Fusion - Batch Size Experiments:

1) Batch Size 4: Test Accuracy: % 22.83 Validation Accuracy: % 18.91 (Figure 29, Figure 30)

2) Batch Size 8: Test Accuracy: % 19.79 Validation Accuracy: % 19.24 (Figure 31, Figure 32)

3) Batch Size 16: Test Accuracy: % 21.18 Validation Accuracy: % 17.45 (Figure 33, Figure 34)

4) Batch Size 32: Test Accuracy: % 13.99 Validation Accuracy: % 15.80 (Figure 35, Figure 36)

In the second stage, I tested the learning rate for 4 different models. For this, I kept the values of batch size: 16, epoch: 10 constant.

Spatial ConvNET - Learning Rate Experiments:

1) Learning Rate 0.00001: Test Accuracy: % 19.10 Validation Accuracy: % 14.74 (Figure 37, Figure 38)

2) Learning Rate 0.00005: Test Accuracy: % 17.36 Validation Accuracy: % 11.16 (Figure 39, Figure 40)

3) Learning Rate 0.0001 Test Accuracy: % 11.46 Validation Accuracy: % 9.29 (Figure 41, Figure 42)

4) Learning Rate 0.0005: Test Accuracy: % 9.38 Validation Accuracy: % 8.19 (Figure 43, Figure 44)

Temporal ConvNET - Learning Rate Experiments:

1) Learning Rate 0.00001: Test Accuracy: % 24.64 Validation Accuracy: % 23.52 (Figure 45, Figure 46)

```
[ ]  class CNN_Spatial(nn.Module):
        def __init__(self):
            super(CNN_Spatial, self).__init__()
            self.layer1 = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.layer2 = nn.Sequential(
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(200704, 1000)
            self.fc2 = nn.Linear(1000, 10)

        def forward(self, x):
            out = self.layer1(x)
            out = self.layer2(out)
            out = out.reshape(out.size(0), -1)
            out = self.drop_out(out)
            out = self.fc1(out)
            out = self.fc2(out)
            return out
```

```
[ ]  class CNN_Temporal(nn.Module):
        def __init__(self):
            super(CNN_Temporal, self).__init__()
            self.layer1 = nn.Sequential(
                nn.Conv2d(18, 32, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.layer2 = nn.Sequential(
                nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2))
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(200704, 1000)
            self.fc2 = nn.Linear(1000, 10)

        def forward(self, x):
            out = self.layer1(x)
            out = self.layer2(out)
            out = out.reshape(out.size(0), -1)
            out = self.drop_out(out)
            out = self.fc1(out)
            out = self.fc2(out)
            return out
```

Figure 4: Late Fusion Model

2) Learning Rate 0.00005: Test Accuracy: % 25.69 Validation Accuracy: % 22.53 (Figure 47, Figure 48)

3) Learning Rate 0.0001: Test Accuracy: % 29.51 Validation Accuracy: % 19.59 (Figure 49, Figure 50)

4) Learning Rate 0.0005: Test Accuracy: % 13.89 Validation Accuracy: % 15.38 (Figure 51, Figure 52)

Early Fusion - Learning Rate Experiments:

1) Learning Rate 0.00001: Test Accuracy: % 27.08 Validation Accuracy: % 20.71 (Figure 53, Figure 54)

2) Learning Rate 0.00005: Test Accuracy: % 23.96 Validation Accuracy: % 22.39 (Figure 55, Figure 56)

3) Learning Rate 0.0001: Test Accuracy: % 28.12 Validation Accuracy: % 19.46 (Figure 57, Figure 58)

4) Learning Rate 0.0005: Test Accuracy: % 14.58 Validation Accuracy: % 17.46 (Figure 59, Figure 60)

Late Fusion - Learning Rate Experiments:

1) Learning Rate 0.00001: Test Accuracy: % 21.18 Validation Accuracy: % 17.45 (Figure 61, Figure 62)

2) Learning Rate 0.00005: Test Accuracy: % 20.49 Validation Accuracy: % 17.65 (Figure 63, Figure 64)

3) Learning Rate 0.0001: Test Accuracy: % 17.36 Validation Accuracy: % 19.09 (Figure 65, Figure 66)

4) Learning Rate 0.0005: Test Accuracy: % 14.24 Validation Accuracy: % 19.17 (Figure 67, Figure 68)

- **Experimental Results**

  When we examine the experiments, we can draw the following conclusions. The Spatial ConvNET model is not enough to actually learn. This model is not from the source but from the data. Because a movement cannot be learned from a single frame. Nevertheless, the highest test I received was 19.10 as a result of

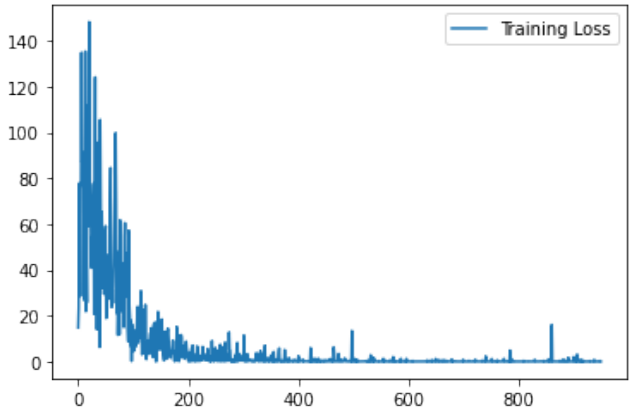

Figure 5: Spatial ConvNET, Batch Size: 4, Train Loss

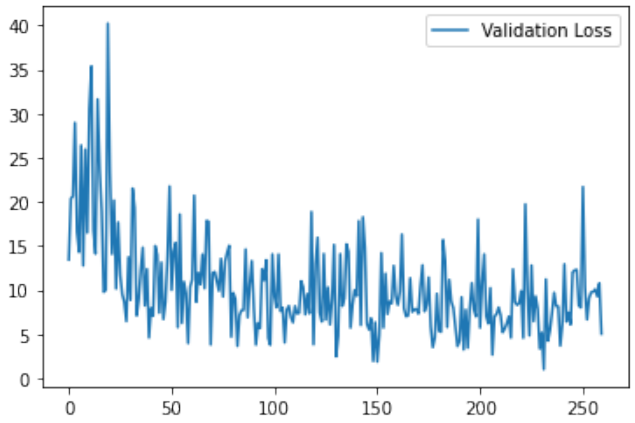

Figure 6: Spatial ConvNET, Batch Size: 4, Validation Loss

the tests I did. The chance factor is also effective in this high. In Temporal ConvNET, we see that the results have improved. Because the machine is now learning the movements, with the flows. When we examine the results, we see that the best result is 29.51 with 16 batch size, 10 epoch, 0.0001 learning rate parameters. A test of 29.51 indicates that the machine has learned the movements. Early Fusion is the model with good results. We see that the accuracy of the results, which we examined in order, has improved. We see that the highest test accuracy is 35.87 with 20 epoch, 4 batch size, 0.00001 learning rate parameters. I can say that it is a really high accuracy for such a limited dataset. We can say that using frame with flows after Early Fusion increases the learning of the machine. When we examine the late fusion, we see that accuracy decreases compared to early fusion. The highest accuracy I received is 29.51. When we examine the results, we see that the results are close with Temporal ConvNET. In general, the approach of the models to the parameters was similar. I got the best results with low batch size, high epoch, low learning rate parameters. The parameters of the Best Model are shown below, and the confussion matrix can be seen in figure 69:

Model: Early Fusion

Epoch: 10

Batch Size: 4

Learning Rate: 0.0001

Test Accuracy: 35.87

Validation Accuracy: 23.64

- **Conclusion**

  The biggest weakness of the code is actually memory management. Since I did my experiments with 25 GB of RAM, I did not have this problem, but this problem can be experienced on a different device. I kept all the images on the disk in RAM to gain speed. Instead, I could open these images when I just

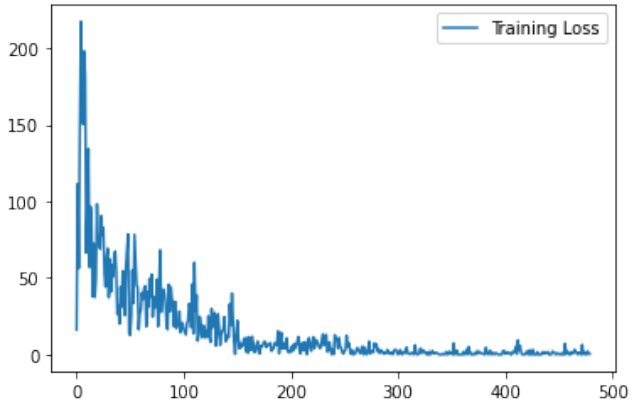

Figure 7: Spatial ConvNET, Batch Size: 8, Train Loss

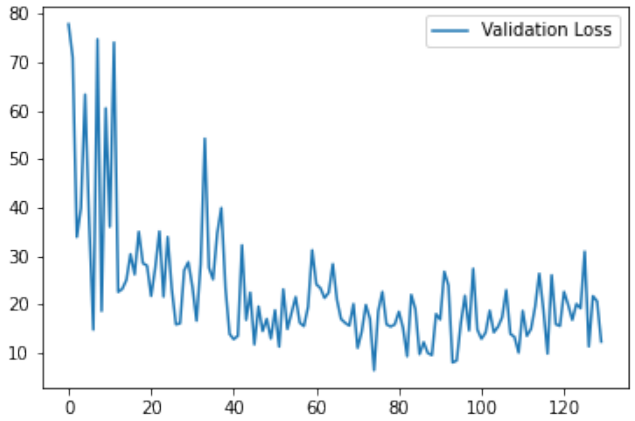

Figure 8: Spatial ConvNET, Batch Size: 8, Validation Loss

kept the path of the images and received them as batch. In this way, I would save RAM. I wrote the code to separate python files because it is understandable, but this means that the same process works separately. Instead, I could merge python files and produce a single file.

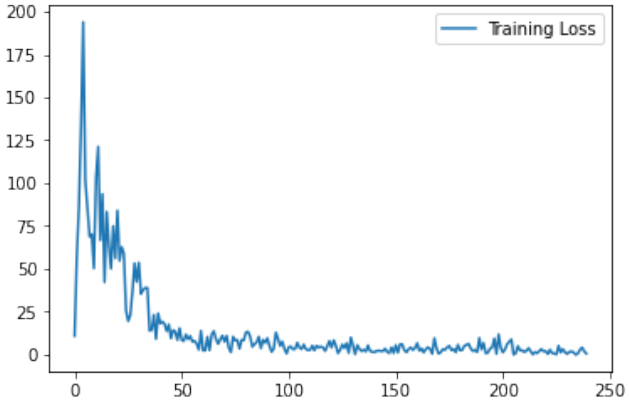

Figure 9: Spatial ConvNET, Batch Size: 16, Train Loss

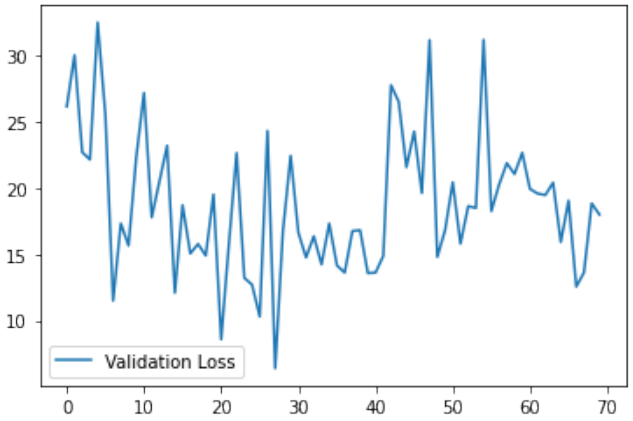

Figure 10: Spatial ConvNET, Batch Size: 16, Validation Loss

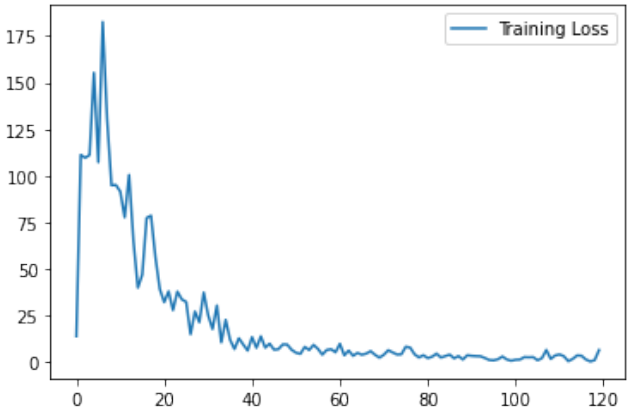

Figure 11: Spatial ConvNET, Batch Size: 32, Train Loss

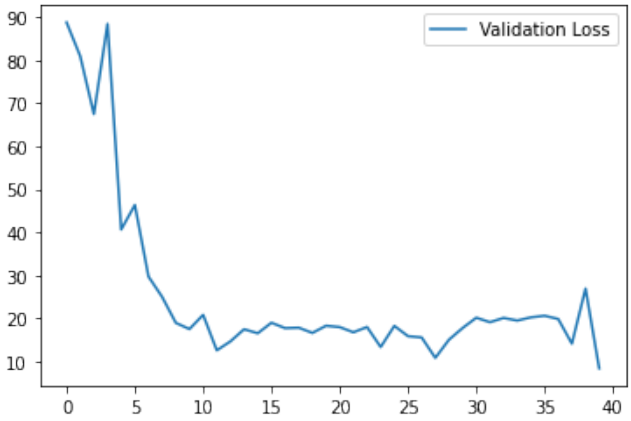

Figure 12: Spatial ConvNET, Batch Size: 32, Validation Loss

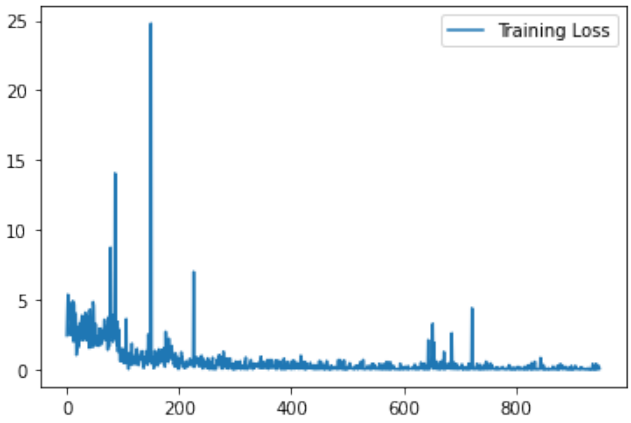

Figure 13: Temporal ConvNET, Batch Size: 4, Train Loss

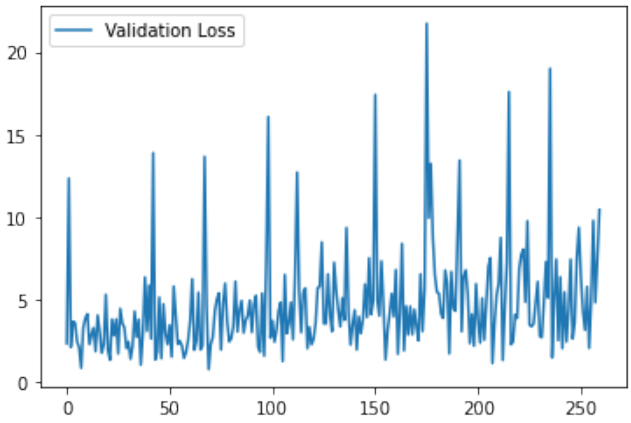

Figure 14: Temporal ConvNET, Batch Size: 4, Validation Loss

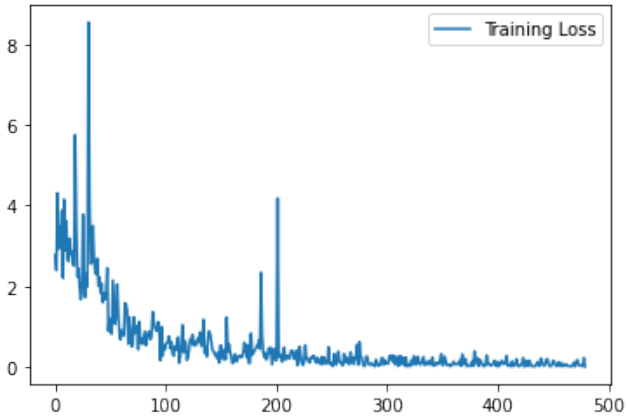

Figure 15: Temporal ConvNET, Batch Size: 8, Train Loss

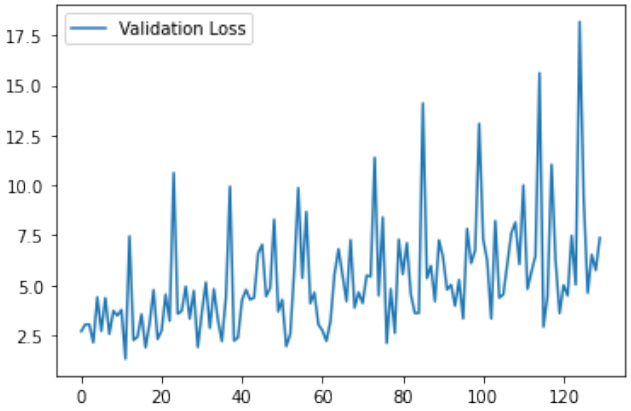

Figure 16: Temporal ConvNET, Batch Size: 8, Validation Loss

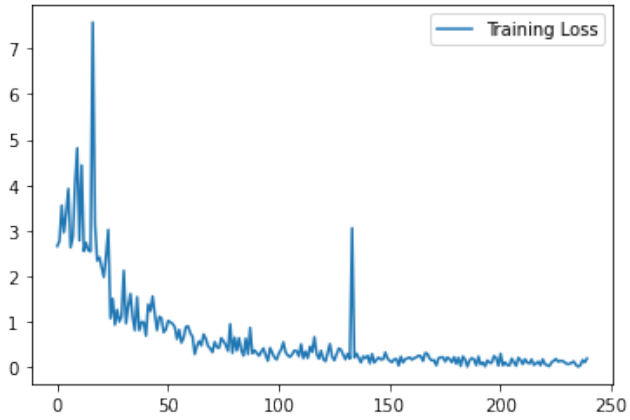

Figure 17: Temporal ConvNET, Batch Size: 16, Train Loss

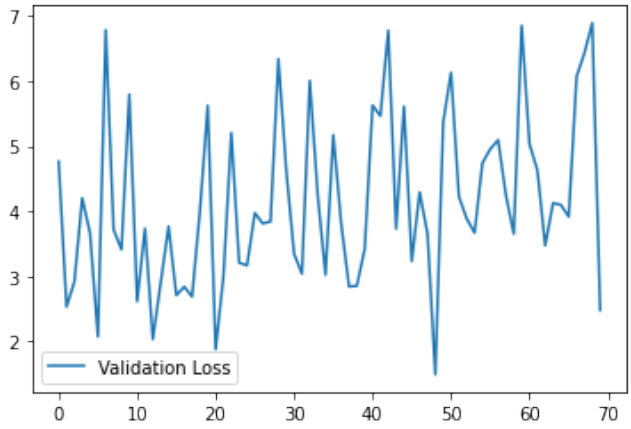

Figure 18: Temporal ConvNET, Batch Size: 16, Validation Loss

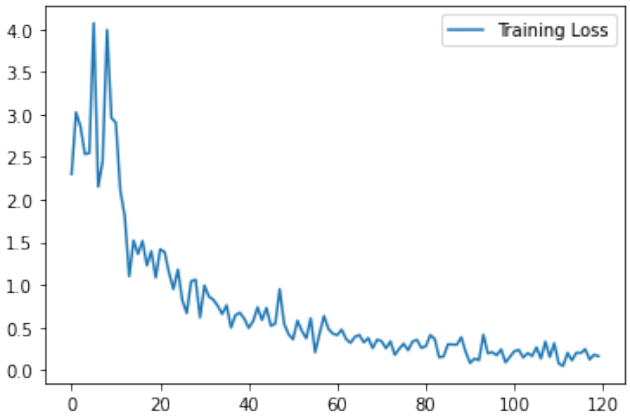

Figure 19: Temporal ConvNET, Batch Size: 32, Train Loss

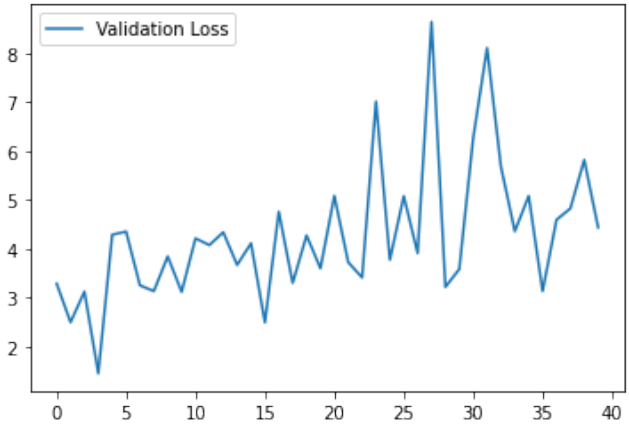

Figure 20: Temporal ConvNET, Batch Size: 32, Validation Loss

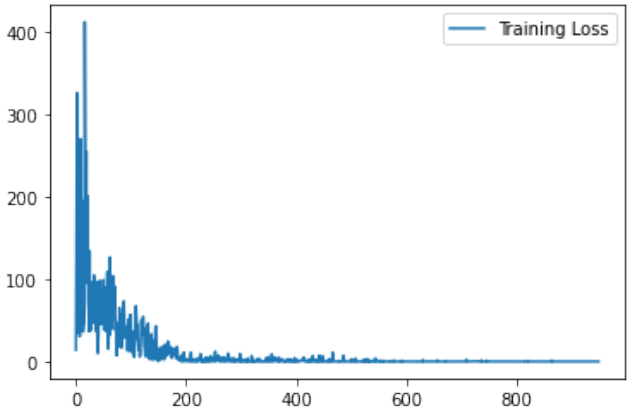

Figure 21: Early Fusion, Batch Size: 4, Train Loss

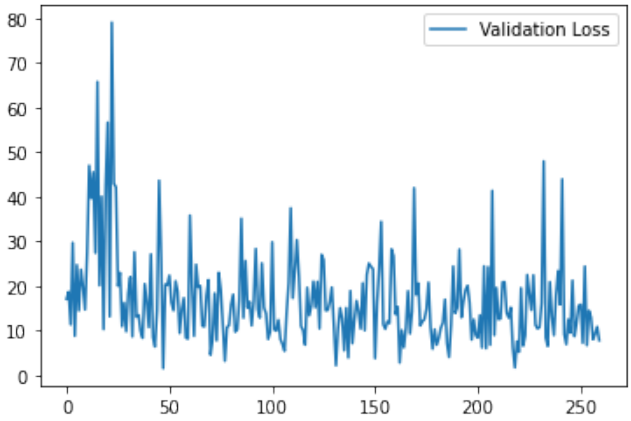

Figure 22: Early Fusion, Batch Size: 4, Validation Loss

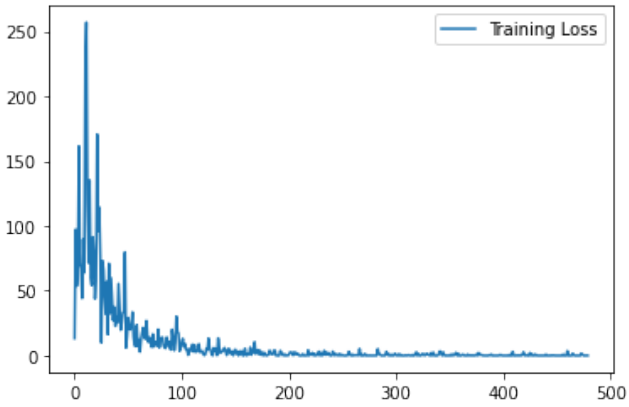

Figure 23: Early Fusion, Batch Size: 8, Train Loss

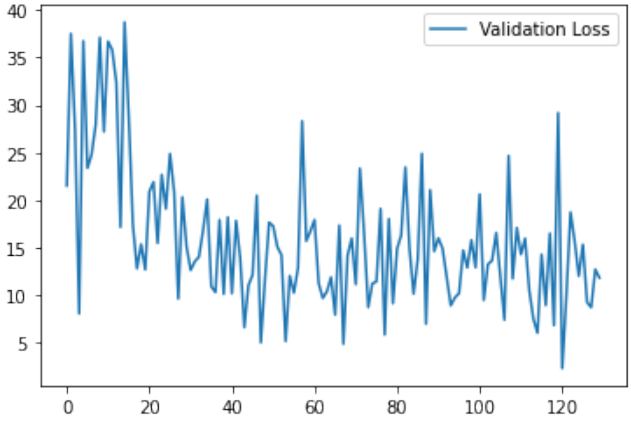

Figure 24: Early Fusion, Batch Size: 8, Validation Loss

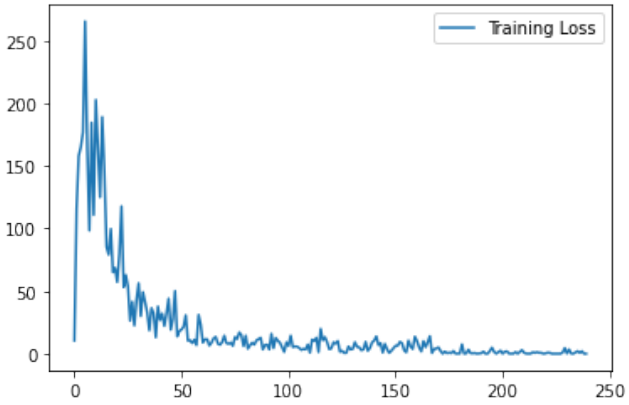

Figure 25: Early Fusion, Batch Size: 16, Train Loss

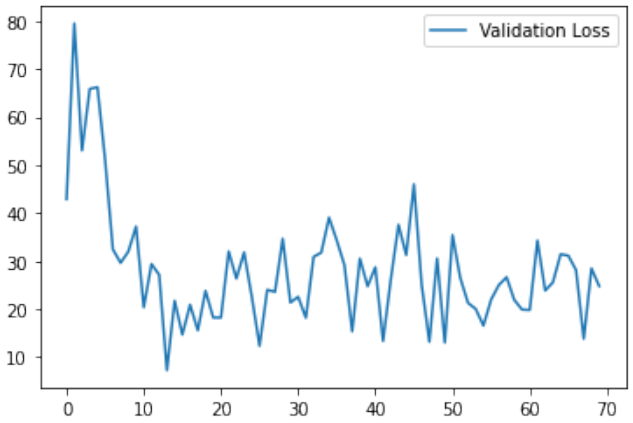

Figure 26: Early Fusion, Batch Size: 16, Validation Loss

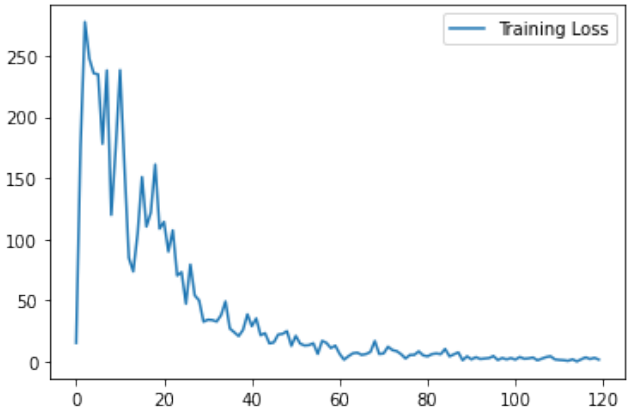

Figure 27: Early Fusion, Batch Size: 32, Train Loss

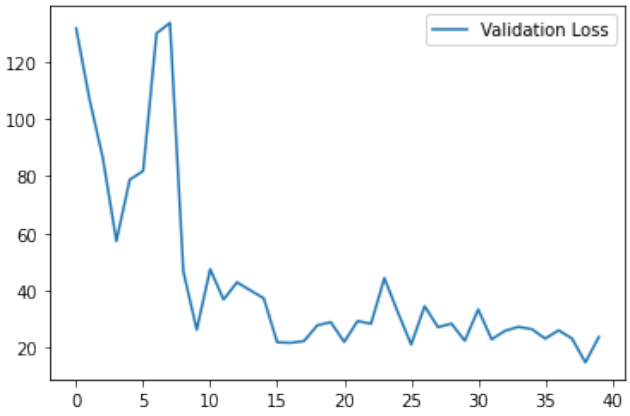

Figure 28: Early Fusion, Batch Size: 32, Validation Loss

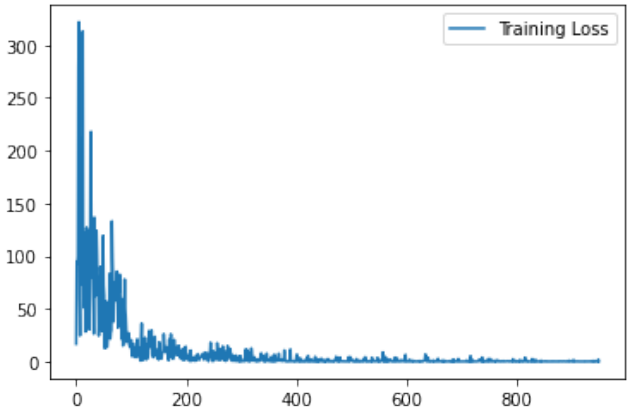

Figure 29: Late Fusion, Batch Size: 4, Train Loss

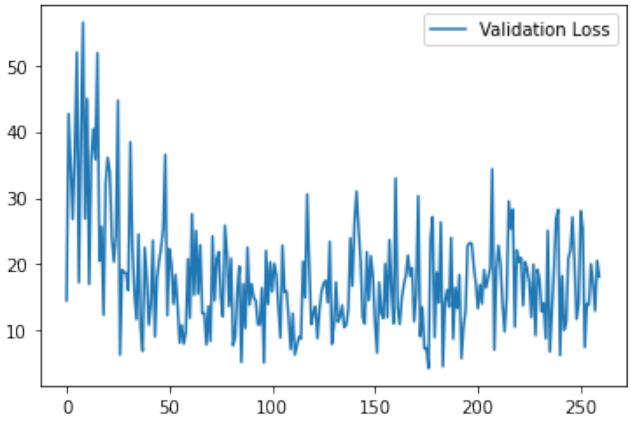

Figure 30: Late Fusion, Batch Size: 4, Validation Loss

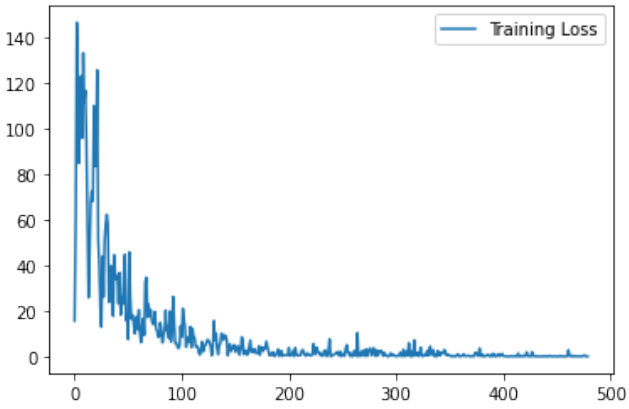

Figure 31: Late Fusion, Batch Size: 8, Train Loss

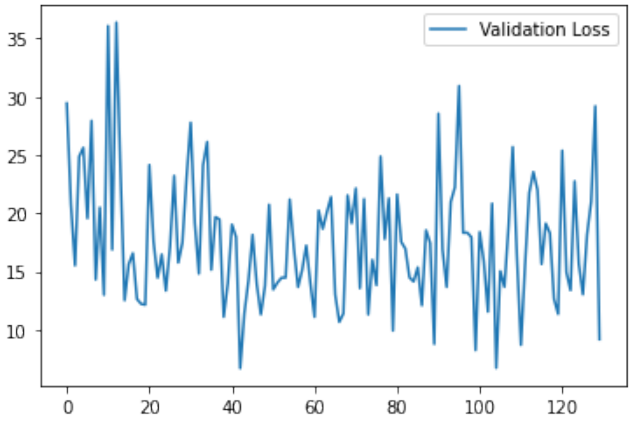

Figure 32: Late Fusion, Batch Size: 8, Validation Loss

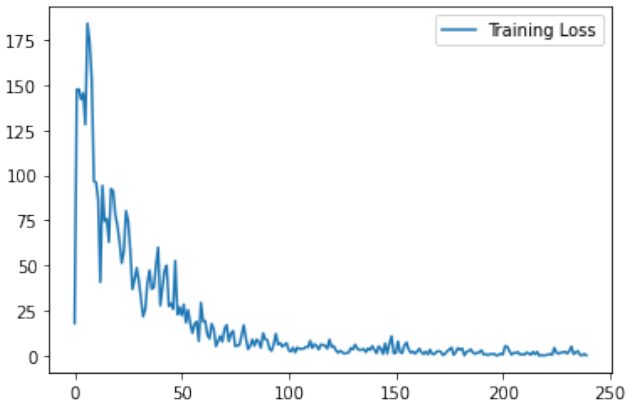

Figure 33: Late Fusion, Batch Size: 16, Train Loss

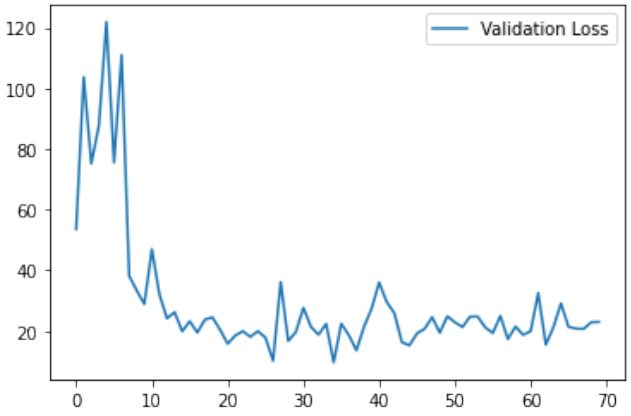

Figure 34: Late Fusion, Batch Size: 16, Validation Loss

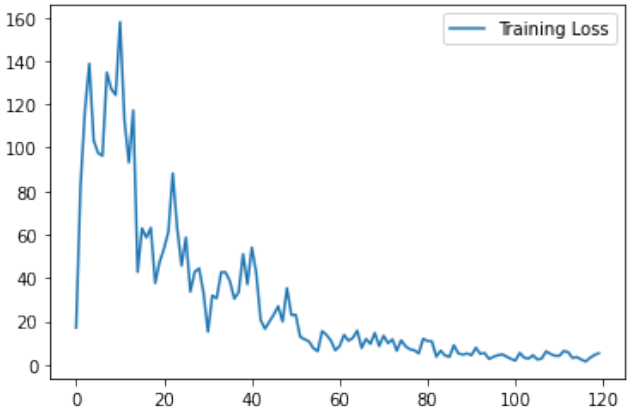

Figure 35: Late Fusion, Batch Size: 32, Train Loss

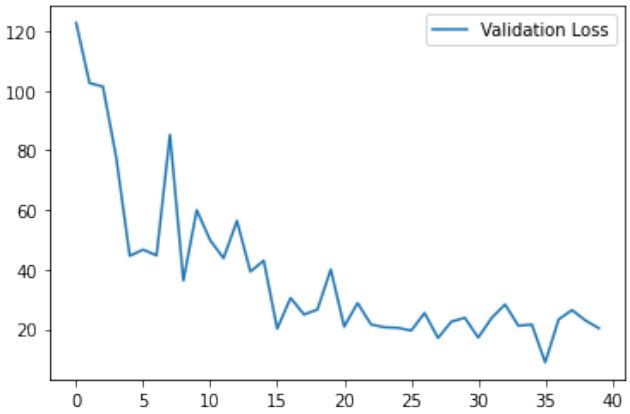

Figure 36: Late Fusion, Batch Size: 32, Validation Loss

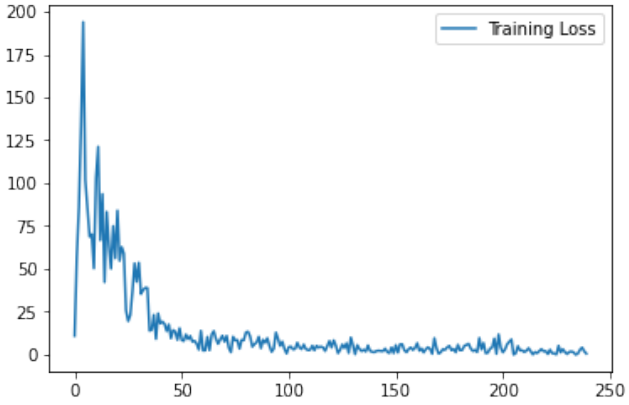

Figure 37: Spatial ConvNET, Learning Rate: 0.00001, Train Loss

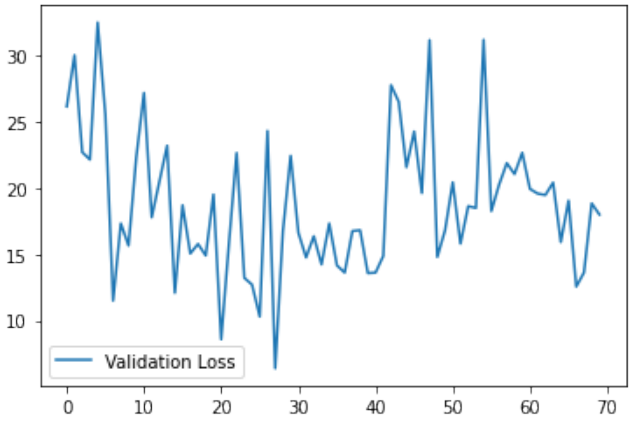

Figure 38: Spatial ConvNET, Learning Rate: 0.00001, Validation Loss

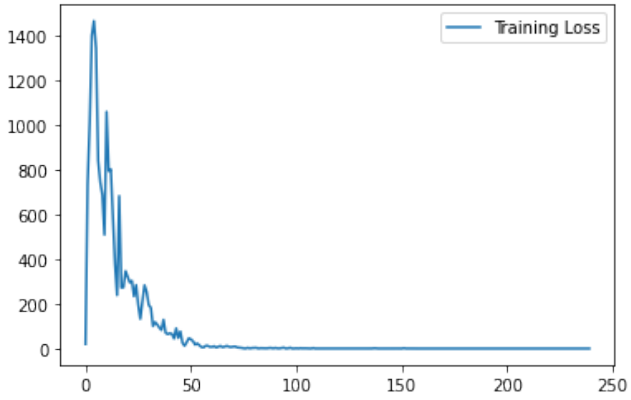

Figure 39: Spatial ConvNET, Learning Rate: 0.00005, Train Loss

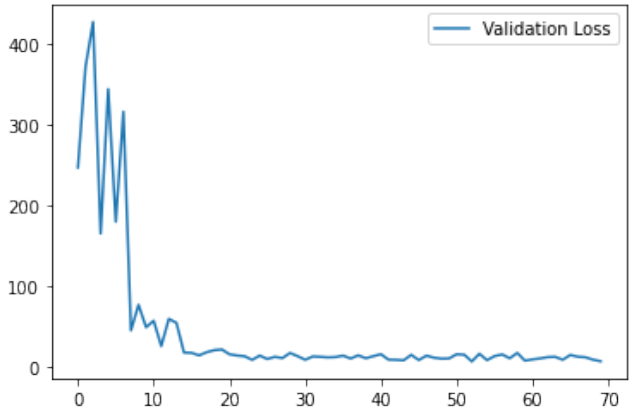

Figure 40: Spatial ConvNET, Learning Rate: 0.00005, Validation Loss

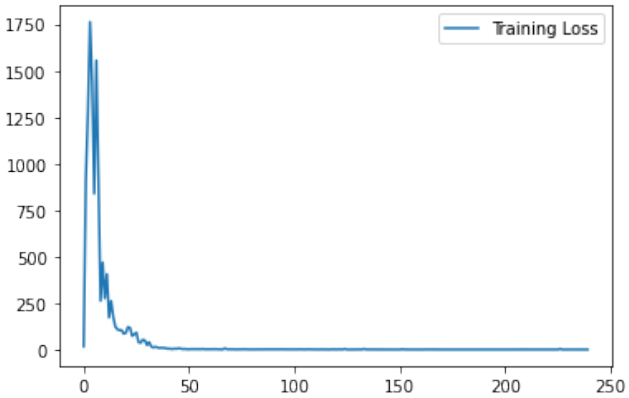

Figure 41: Spatial ConvNET, Learning Rate: 0.0001, Train Loss

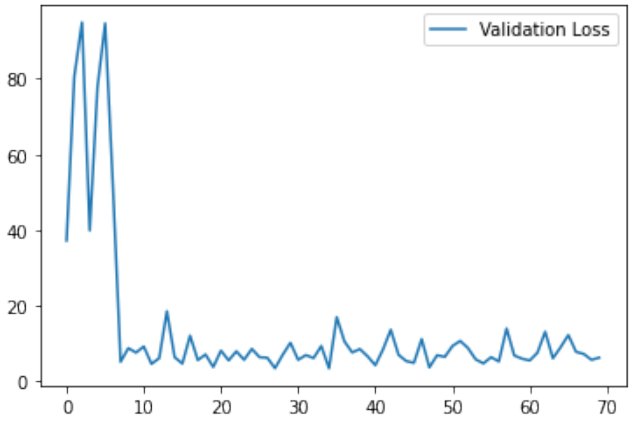

Figure 42: Spatial ConvNET, Learning Rate: 0.0001, Validation Loss

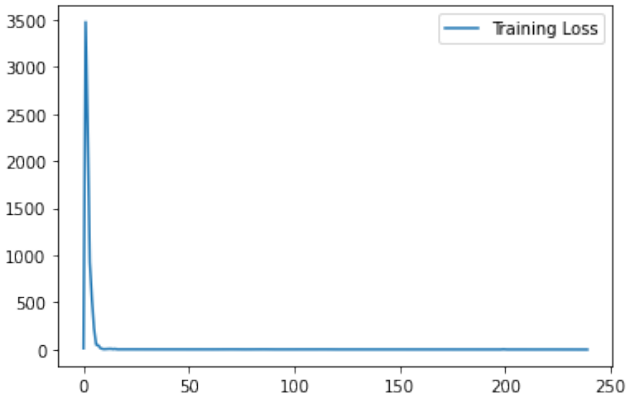

Figure 43: Spatial ConvNET, Learning Rate: 0.0005, Train Loss

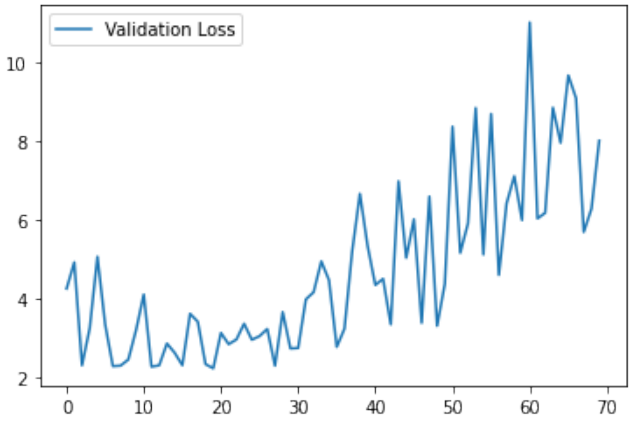

Figure 44: Spatial ConvNET, Learning Rate: 0.0005, Validation Loss

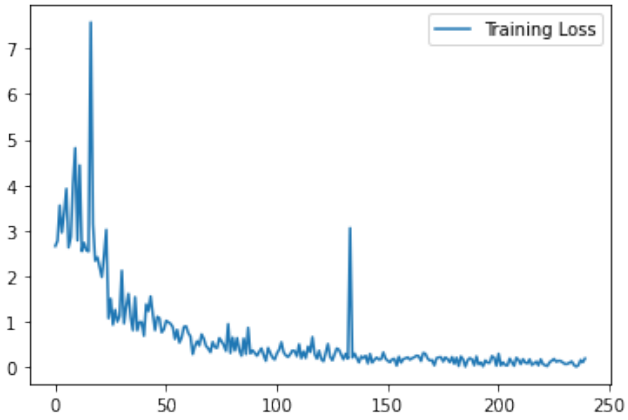

Figure 45: Temporal ConvNET, Learning Rate: 0.00001, Train Loss

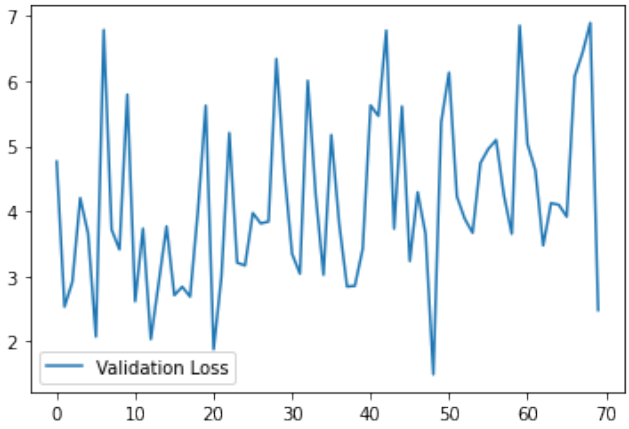

Figure 46: Temporal ConvNET, Learning Rate: 0.00001, Validation Loss

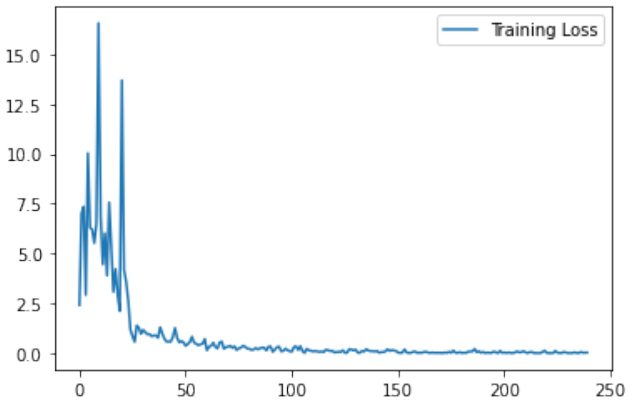

Figure 47: Temporal ConvNET, Learning Rate: 0.00005, Train Loss

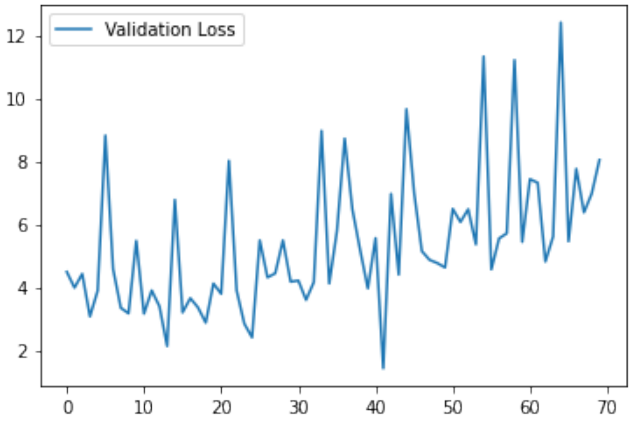

Figure 48: Temporal ConvNET, Learning Rate: 0.00005, Validation Loss

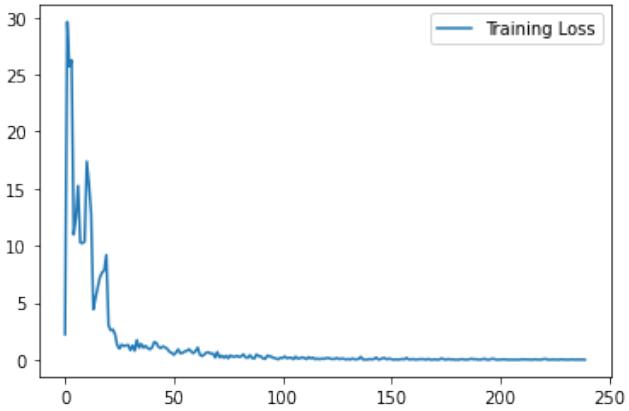

Figure 49: Temporal ConvNET, Learning Rate: 0.0001, Train Loss

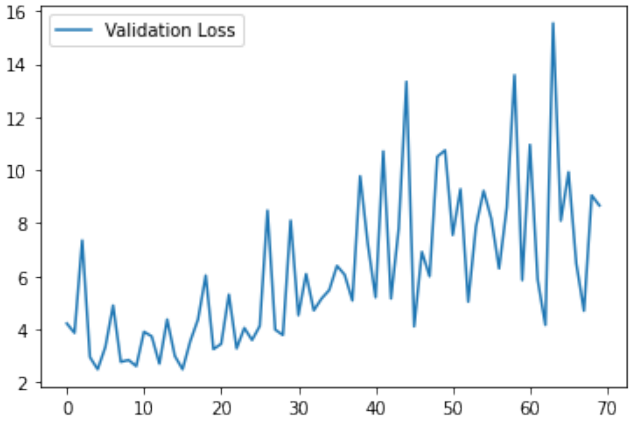

Figure 50: Temporal ConvNET, Learning Rate: 0.0001, Validation Loss

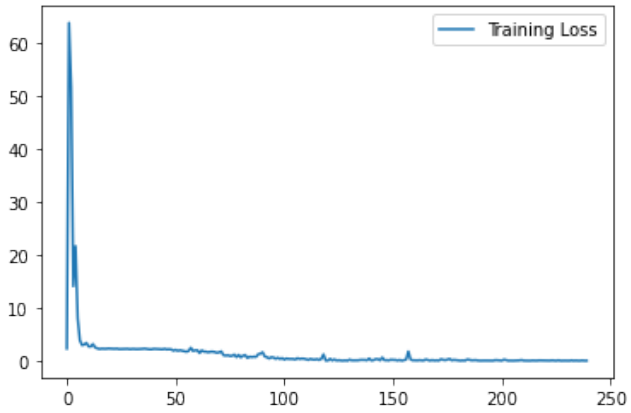

Figure 51: Temporal ConvNET, Learning Rate: 0.0005, Train Loss

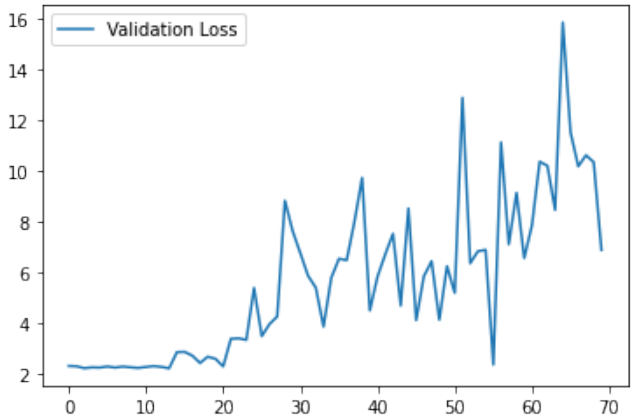

Figure 52: Temporal ConvNET, Learning Rate: 0.0005, Validation Loss

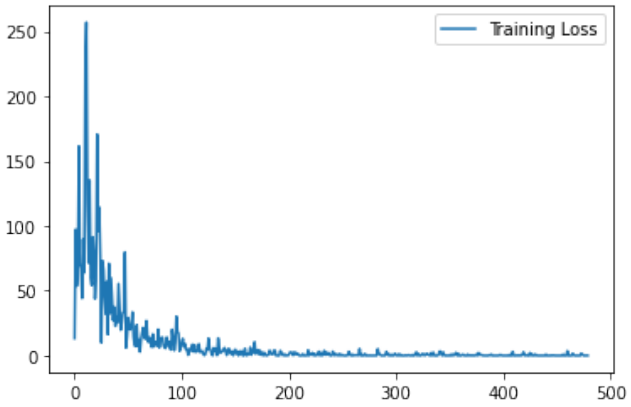

Figure 53: Early Fusion, Learning Rate: 0.00001, Train Loss

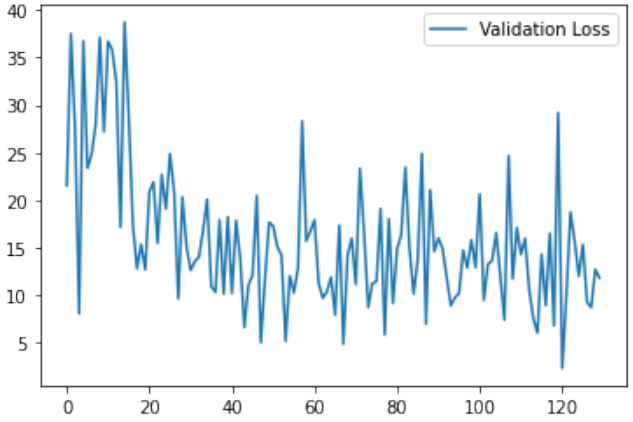

Figure 54: Early Fusion, Learning Rate: 0.00001, Validation Loss

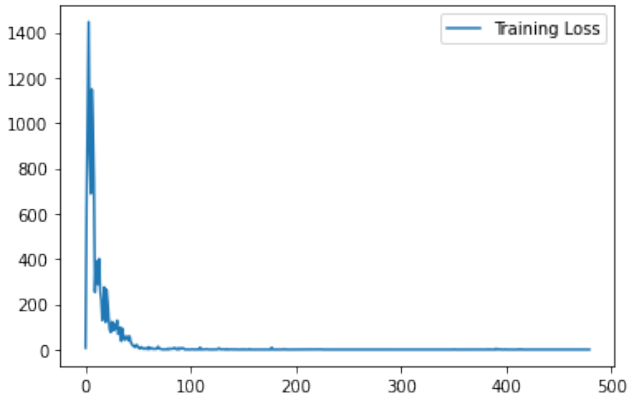

Figure 55: Early Fusion, Learning Rate: 0.00005, Train Loss

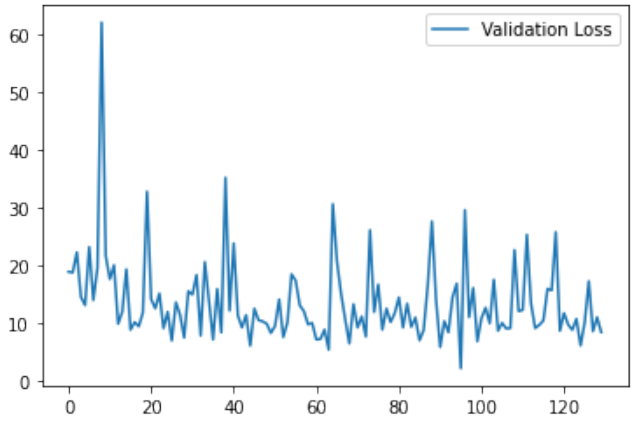

Figure 56: Early Fusion, Learning Rate: 0.00005, Validation Loss

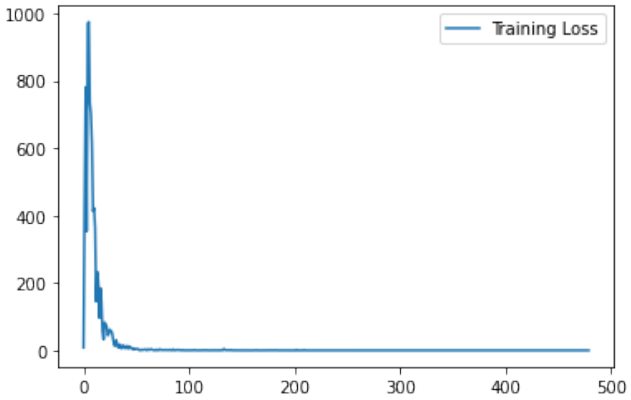

Figure 57: Early Fusion, Learning Rate: 0.0001, Train Loss

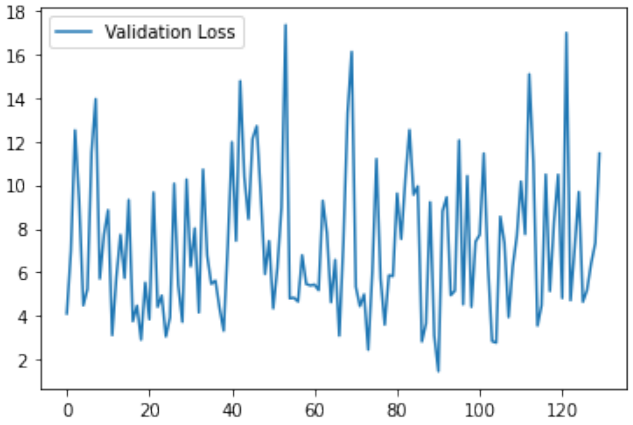

Figure 58: Early Fusion, Learning Rate: 0.0001, Validation Loss

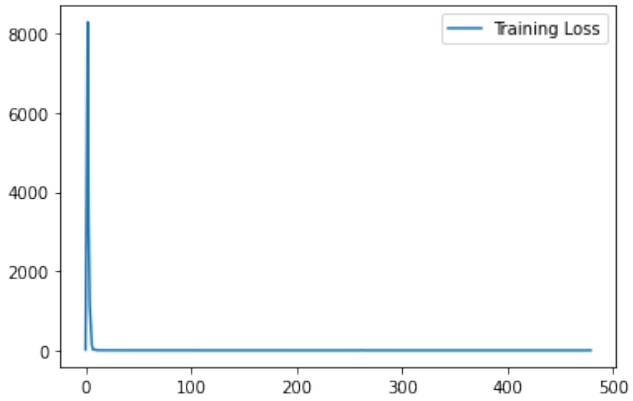

Figure 59: Early Fusion, Learning Rate: 0.0005, Train Loss

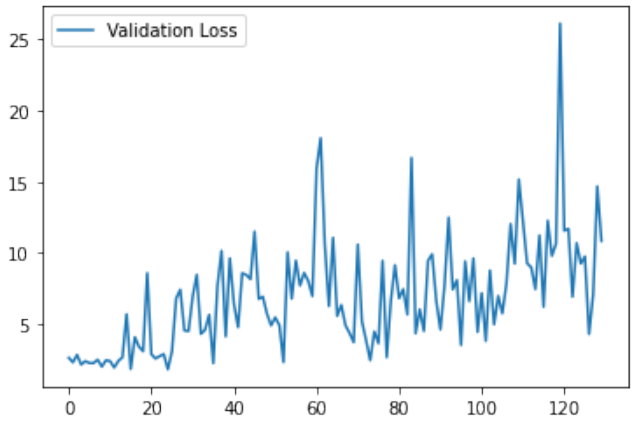

Figure 60: Early Fusion, Learning Rate: 0.0005, Validation Loss

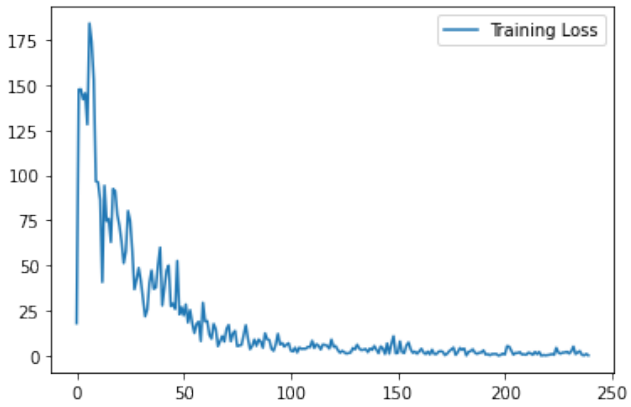

Figure 61: Late Fusion, Learning Rate: 0.00001, Train Loss

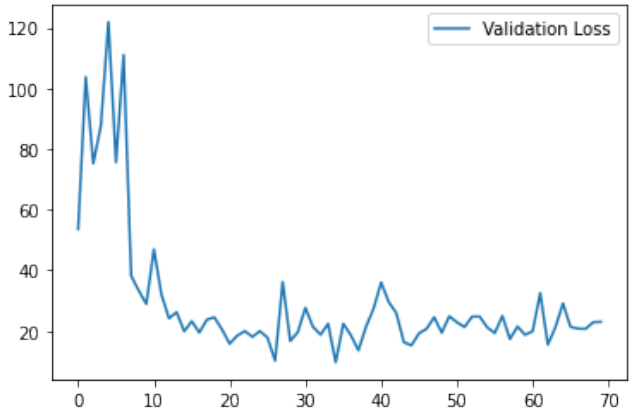

Figure 62: Late Fusion, Learning Rate: 0.00001, Validation Loss

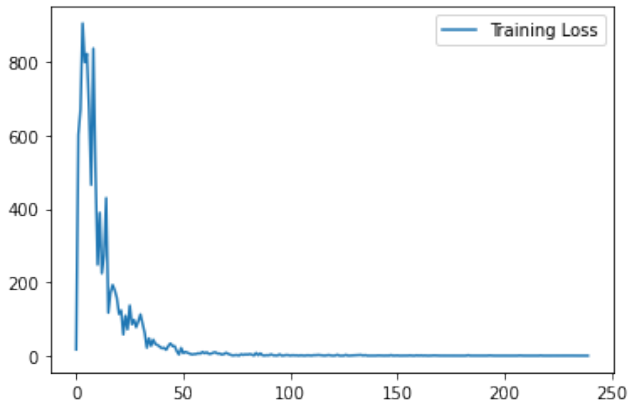

Figure 63: Late Fusion, Learning Rate: 0.00005, Train Loss

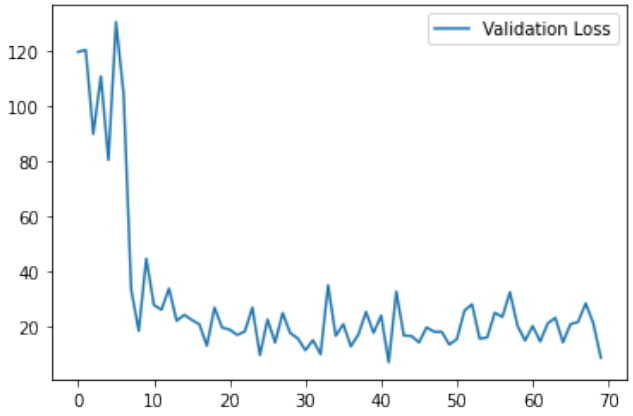

Figure 64: Late Fusion, Learning Rate: 0.00005, Validation Loss

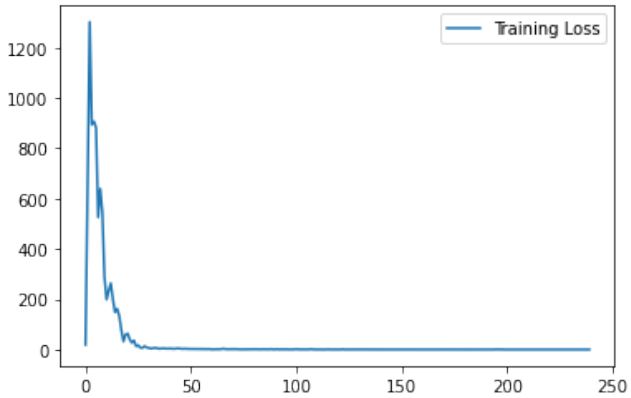

Figure 65: Late Fusion, Learning Rate: 0.0001, Train Loss

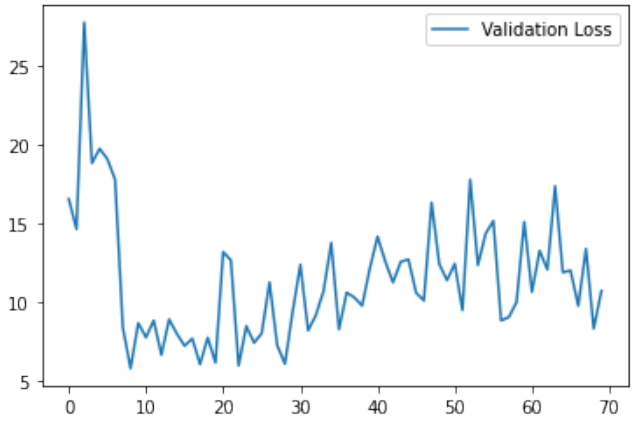

Figure 66: Late Fusion, Learning Rate: 0.0001, Validation Loss

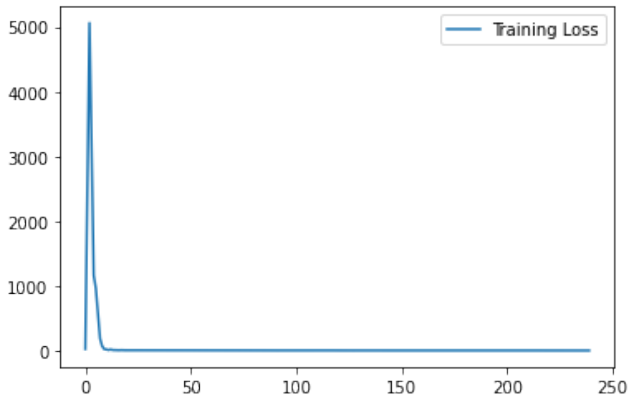

Figure 67: Late Fusion, Learning Rate: 0.0005, Train Loss

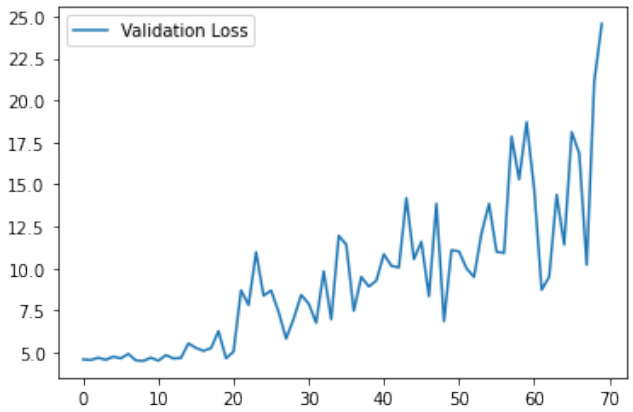

Figure 68: Late Fusion, Learning Rate: 0.0005, Validation Loss

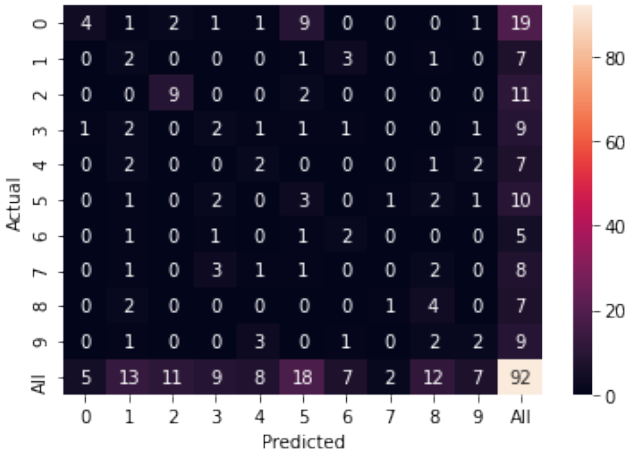

Figure 69: Confussion Matrix