DEEP LEARNING PROJECT REPORT: $C3D_{small}$ FOR UCF11 VIDEO CLASSIFICATION

A PREPRINT

Xiao Jiang
Department of Computer Science
Rensselaer Polytechnic Insitute
Troy, NY 12180
`jiangx5@rpi.edu`

May 5, 2019

ABSTRACT

In this programming assignment, a 3-D convolution neural network is used to train and classify video frames in UCF YouTube Action Dataset (UCF11). The training converges in approximately 15 ephochs of training data (6000 sequences) and the final prediction of classification is close to expected performance of >80

1 Theory of Convolutional Neural Network

1.1 Forward Propagation

1.2 Input layer

The input X is usually a 2-D or multi-dimensional matrix representing an image or pre-processed data. The forward prop can be shown below.

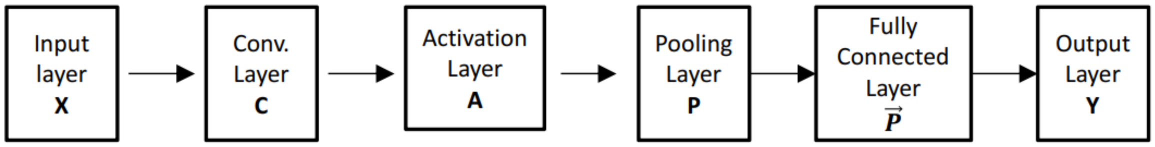


Figure 1: Diagram for a typical CNN

1.2.1 Convolution layer

The convolution layer that is a combination of various feature layers can be calculated as shown below.

1.2.2 Activation layer

The activation layer, with $ReLU$ activation function is simply applying ReLU to each cell of convolution layer:

$$\mathbf{A} = ReLU(\mathbf{C}) \tag{1}$$

Let $\mathbf{X}^{\mathrm{MxN\times D}}$ be an input image, with D channels

Let $\mathbf{W}^{\mathrm{x}^{\mathrm{KxK\times D}}}$ be the filter and $\mathbf{W}_0^{\mathrm{x}}$ be the bias matrix

Let $\mathbf{C}^{N_r^c \times N_c^c}$ be the convolutional layer, where N_r^c and N_c^c are its

height and width, and $N_r^c = \frac{\mathrm{M - K}}{\mathrm{s}} + 1$, $N_c^c = \frac{\mathrm{N - K}}{\mathrm{s}} + 1$

assuming stride of s without zero - padding

For r = 1 to N_r^c

For c = 1 to N_c^c

$$\mathbf{C}[r][c] = \sum_{l=1}^{D}\sum_{i=1}^{K}\sum_{j=1}^{K}\mathbf{X}[(r-1)\times s+i][(c-1)\times s+j][l]\mathbf{W}^x[i][j][l] + \mathbf{W}_0^x[r][c]$$

Figure 2: From input layer to convolutional feature layer

1.2.3 Pooling layer

The pooling layer tries to shrink the activation layer, and in this final project we use max pooling. For any stride, the maximum among cells covered by the sliding window is the 'candidate' for the pooling layer $\mathbf{P}$

1.2.4 Fully-connected layer

While the pooling layer can be of arbitrary shape, the fully connected layer is the flattened pooling layer.

$$\bar{\mathbf{P}} = flatten(\mathbf{P}) \tag{2}$$

1.2.5 Output Layer

The output layer is similar to regression, and is already covered in other PA write-ups

1.3 Back-Prop

The back propagation procedures can be shown below

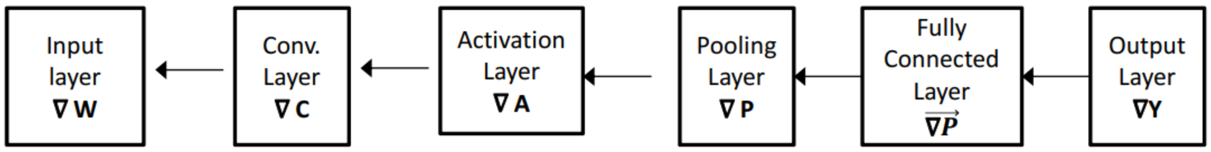


Figure 3: From output layer to input layer for gradient calculation

1.3.1 Gradient of output layer

The output layer's gradient is highly dependent on the loss function $l()$. For a mean-squared loss, the gradient of output layer can be calculated as shown below.

1.3.2 Gradient of FC layer weights

The gradient of weight for output layer and fully connected layer is the same as deep neural network, as shown in the image below

$$\mathrm{l}(\mathbf{t}[\mathrm{m}],\, \hat{\mathbf{y}}[\mathrm{m}]) = \frac{1}{2}\, (\mathbf{t}[\mathrm{m}]-\hat{\mathbf{y}}[\mathrm{m}])^{\mathrm{t}}(\mathbf{t}[\mathrm{m}]-\hat{\mathbf{y}}[\mathrm{m}])$$

$$\begin{aligned}\nabla\hat{\mathbf{y}}[m] &= \frac{1}{2}\frac{\partial(\mathbf{t}[m]-\hat{\mathbf{y}}[m])^t(\mathbf{t}[m]-\hat{\mathbf{y}}[m])}{\partial\hat{\mathbf{y}}} \\ &= -(\mathbf{t}[m]-\hat{\mathbf{y}}[m])\end{aligned}$$

Figure 4: Gradient of output for mean squared loss

$$\begin{aligned}&\hat{\mathbf{y}} = g((\mathbf{W}^o)^t\vec{\mathbf{P}} + \mathbf{W}_0^o) \\ &\nabla\mathbf{W}^o = \frac{\partial\hat{\mathbf{y}}}{\partial\mathbf{W}^o}\nabla\hat{\mathbf{y}} \qquad \nabla\mathbf{W}_0^o = \frac{\partial\hat{\mathbf{y}}}{\partial\mathbf{W}_0^o}\nabla\hat{\mathbf{y}} \\ &\nabla\vec{\mathbf{P}} = \frac{\partial\hat{\mathbf{y}}}{\partial\vec{\mathbf{P}}}\nabla\hat{\mathbf{y}}\end{aligned}$$

Figure 5: Gradient of output for output loss and full connnected layer

1.4 Gradient of pooling layer

The gradient of pooling layer, $\nabla\mathbf{P}$ is trivial.

$$\nabla\mathbf{P} = unflatten(\nabla\bar{\mathbf{P}}) \tag{3}$$

1.5 Gradient of activation layer

The gradient of activation layer is a function of the gradient of pooling layer and the pooling layer itself.

$$\begin{aligned}\mathbf{P}[r][c] &= \max_{\substack{1\le i\le d\\ 1\le j\le d}} \mathbf{A}[(r-1)s+i][(c-1)s+j] \qquad (1) \\ &= \max_{\substack{(r-1)s+1\le i'\le(r-1)s+d\\ (c-1)s+1\le j'\le(c-1)s+d}} \mathbf{A}[i'][j'] \quad (1)\end{aligned}$$

$$\text{where } \mathrm{i}' = (r-1)s+i \qquad \mathrm{j}' = (c-1)s+j$$

$$i^*, j^* = \mathop{\arg\max}_{\substack{(r-1)s+1\le i'\le(r-1)s+d\\ (c-1)s+1\le j'\le(c-1)s+d}} \mathbf{A}[i'][j']$$

$$\nabla\mathbf{A} = \frac{\partial\mathbf{P}}{\partial\mathbf{A}}\nabla\mathbf{P} = \sum_{r=1}^{N_r^P}\frac{\partial\mathbf{P}[r]}{\partial\mathbf{A}}\nabla\mathbf{P}[r] = \sum_{r=1}^{N_r^P}\sum_{c=1}^{N_c^P}\frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}}\nabla\mathbf{P}[r][c]$$

$$= \sum_{r=1}^{N_r^P}\sum_{c=1}^{N_c^P}\begin{bmatrix} \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[1][1]} & \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[1][2]} \cdots \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[1][N_c^A]} \\ \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[2][1]} & \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[2][2]} \cdots \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[1][N_c^A]} \\ \cdots & \\ \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[N_r^A][1]} & \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[N_r^A][2]} \cdots \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[N_r^A][N_c^A]} \end{bmatrix}\nabla\mathbf{P}[r][c]$$

$$\text{using Eq (1)},\ \frac{\partial\mathbf{P}[r][c]}{\partial\mathbf{A}[i'][j']} = \begin{cases}1 & \text{if } \mathrm{i}' = \mathrm{i}^* \text{ and } \mathrm{j}' = \mathrm{j}^* \\ 0 & \text{else}\end{cases}$$

Figure 6: Gradient of activation layer

1.6 Gradient of convolutional layer

The gradient of a convolutional layer can be expressed as below. It is a function of the gradient of activation layer and the value of convluational layer.

$$
\begin{aligned}
&\mathbf{A}[r][c] = \mathrm{Re}Lu(\mathbf{C}[r][c]) \\
&\nabla\mathbf{C} = \frac{\partial\mathbf{A}}{\partial\mathbf{C}}\nabla\mathbf{A} = \sum_{r=1}^{N_r^a}\sum_{c=1}^{N_c^a}\frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}}\nabla\mathbf{A}[r][c] \\
&= \sum_{r=1}^{N_r^a}\sum_{c=1}^{N_c^a}\begin{bmatrix} \frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}[1][1]} \cdots \frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}[1][N_c^c]} \\ \cdots \\ \frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}[N_r^c][1]} \cdots \frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}[N_r^c][N_c^c]} \end{bmatrix}\nabla\mathbf{A}[r][c] \\
&\text{where} \\
&\frac{\partial\mathbf{A}[r][c]}{\partial\mathbf{C}[i][j]} = \frac{\partial\,\mathrm{Re}LU(C[r][c])}{\partial\mathbf{C}[i][j]} = \begin{cases} 1 \text{ if } \ i=\mathrm{r} \text{ and } \mathrm{j}=\mathrm{c} \text{ and } \mathbf{C}[i][j]>0 \\ 0 \end{cases}
\end{aligned}
$$

Figure 7: Gradient of convolution layer

1.7 Gradient of weights/bias for input layer

The gradient of weights for convolution layer's weights can be calculated from the gradient of convolutional layer itself and value of input layer. So is the bias.

$$
\mathbf{C}[\mathrm{r}][\mathrm{c}] = \sum_{l=1}^{D}\sum_{\mathrm{i}=1}^{\mathrm{K}}\sum_{\mathrm{j}=1}^{\mathrm{K}}\mathbf{W}^X[i][j][l]\mathbf{X}[(r-1)s+i][(c-1)s+j][l] + \mathbf{W}_0^X \qquad (1)
$$

$$
\nabla\mathbf{W}^X = \frac{\partial\mathbf{C}}{\partial\mathbf{W}^X}\nabla\mathbf{C} = \sum_{r=1}^{N_r^C}\sum_{c=1}^{N_c^C}\frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X}\nabla\mathbf{C}[r][c] = \sum_{r=1}^{N_r^C}\sum_{c=1}^{N_c^C}\begin{bmatrix} \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[1][1]} & \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[1][2]} & \cdots & \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[1][K]} \\ \cdots\cdot \\ \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[K][1]} & \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[K][2]} & \cdots & \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[K][K]} \end{bmatrix}\nabla\mathbf{C}[r][c]
$$

$$
\text{Note}\ \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[i][j]} = \begin{bmatrix} \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[i][j][1]} \\ \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[i][j][2]} \\ \cdots \\ \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}^X[i][j][D]} \end{bmatrix}
$$

$$
\text{where using Eq. 1, } \frac{\partial\mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial\mathbf{W}[i][j][l]} = \mathbf{X}[(r-1)s+i][(c-1)s+j][l]
$$

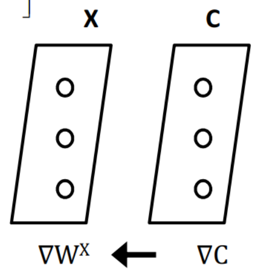


Figure 8: Gradient of conv. weights

1.8 Weight update

For the $K+1$th step of the training, any weight Θ can be update with learning rate η and gradient from the Kth step

$$
\Theta^{K+1} = \Theta^K - \eta * \nabla\Theta^K \qquad (4)
$$

$$\mathbf{C}[\mathrm{r}][\mathrm{c}] = \sum_{l=1}^{D}\sum_{\mathrm{i}=1}^{\mathrm{K}}\sum_{\mathrm{j}=1}^{\mathrm{K}} \mathbf{W}^{X}[i][j][l]\mathbf{X}[(r-1)s+i][(c-1)s+j][l] + \mathbf{W}_0^{X}$$

$$\nabla \mathbf{W}_0^{X} = \frac{\partial \mathbf{C}}{\partial \mathbf{W}_0^{X}} \nabla \mathbf{C} = \sum_{r=1}^{N_r^C}\sum_{c=1}^{N_c^C} \frac{\partial \mathbf{C}[\mathrm{r}][\mathrm{c}]}{\partial \mathbf{W}_0^{X}} \nabla \mathbf{C}[r][c] = \sum_{r=1}^{N_r^C}\sum_{c=1}^{N_c^C} \nabla \mathbf{C}[r][c]$$

Figure 9: Gradient of conv. bias

2 Introduction to C3D model for spatial-temporal convolutional neural network

2.1 Original neural network design

The authors of paper 'Learning Spatiotemporal Features with 3D Convolutional Networks '[1] proposes that performing 3-D convolution over a volume of videoframes is able to extract both spatial and temporal feature of the video. This is justified by the state-of-art performance of the 'C3D' deep CNN on UCF101 test set. The test set is much larger than the one for this final project (101 classes v.s 11 classes). The original neural network incentives and design can be shown in figure 10 and 11.

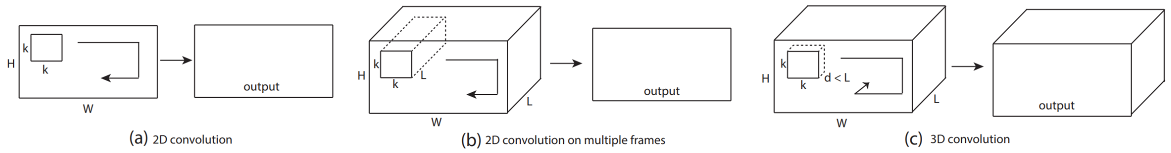


Figure 10: Convolution over a stacked video frame volume generates features in temporal and spatial features

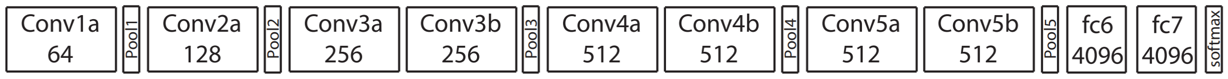


Figure 11: The design of C3D neural network for UCF101 video classification

2.2 Results of original C3D on UCF 101 dataset

The training has 1.3M iterations with 13 epochs of trainning data. The trained model achieved 85.1% accuracy on UCF101 and 85.2% on Sports1M dataset. Such accuracy on large dataset justifies a similar implementation of C3D neural network on UCF11 trainning data for this project.

3 Design choice

The model for parameter training on UCF11 dataset include :

- A sequence of simplified 3D convolutional filters for spatial-temporal feature extraction for video volume formed by stacking a sequence of video frame.
- Two fully connected layers for classification output after 3D convolution

3.1 CNN layout

The CNN is kept simple for a video volume in a video sequence of 30 frames each. The output layer uses softmax function for classification. The configuration for CNN can be shown in table 1.

Layer	Size in	Size out	Kernel	Stride	Activation
conv1	30, 64, 64, 3	30, 64, 64, 32	3 ,3, 3, 3, 32	1	ReLU
max_pool1	30, 64, 64, 32	30, 32, 32, 32	1, 2, 2, 1	2	
conv2	30, 32, 32, 32	28, 30, 30, 64	3, 3, 3, 32, 64	1	ReLU
max_pool2	28, 30, 30, 64	14, 15, 15, 64	2, 2, 2, 1	2	
conv3a	14, 15, 15, 64	10, 11, 11, 128	5, 5, 5, 64, 128	1	ReLU
conv3b	10, 11, 11, 128	6, 7, 7, 128	5, 5, 5, 128, 128	2	ReLU
max_pool3	6, 7, 7, 128	3, 3, 3, 128	1, 2, 2, 2, 1	2	
flatten	3, 3, 3, 128	3456			
fc1	2048				ReLU
fc2	1024				ReLU
output	11				

3.2 Modification made on $C3D_{small}$ for this project

The following modification has been made for C3D model to accommodate faster converge while maintaining accuracy on UCF11 dataset:

- The number of convolution layers is reduced from 5 to 3
- The padding uses 'no-padding' ('VALID') instead of zero-padding ('SAME')
- FC weights are changed accordingly due to the change of output shape, greatly reducing the number of parameters

3.3 Dataset selection

The Data set is split into 3 parts: Training, validation and test. Before the dataset is split, a shuffle is performed on axis 0 so all video sequences are randomized.

3.3.1 Training Set

Out of the 7200 video sequences given, the first 6000 sequences are used for training.

3.3.2 Validation set

Validation set is used during training steps to monitor whether the training is performing well and compare to testing error to determine over-fitting. Here, I choose the 6000-6200th sequences of dataset for validation to ensure both speed and diversity.

3.3.3 Test Set

Though using random data for testing is a good practice since ordered fraction of the whole data set may contain limited types of video frame and thus result in less 'unbiased' test result, I here choose to use the last 1000 out of the 7200 video sequences for testing. The test data set is large and can hopefully cover enough situations.

3.4 Training design choice

3.4.1 Loss function

Cross entropy loss is used for trainning.

With loss for each sequence defined, an $AdamOptimizer$ is used for minimizing the loss by gradient descent for parameter earning.

3.4.2 Regularizer

To punish over-fitting, an l2 regulization is used for the output layer and the value is added to total loss.

```
regularizers = (tf.nn.l2_loss(out_weights) + tf.nn.l2_loss(out_biases))
loss += 5e-4 * regularizers
```

3.4.3 Batch size

I choose a batch size of 10 with an RTX2080 and achieved 60% utilization rate while keep the convergence fast.

3.4.4 Learning rate exponential decay

The starting learning rate η I chose was 0.0001 for a batch size of 10. The learning rate exponentially decay with a factor of 0.9. With around 15 epoches of training, the finishing learning rate is approximately 0.00060788327.

4 Results and Discussions

4.1 Training/Validation Accuracy

The training and validation pixel error from GT through 15 epochs data can be shown below, in figure 12. Is can be shown that the model converges relatively slowly. However, the C3D small model still achieved >90% accuracy on the validation set. The training accuracy is not a good evaluation of performance since the batch size is only 10, the accuracy is always accurate to tenth of a digit. The validation set's accuracy do not diverge much from training accuracy.

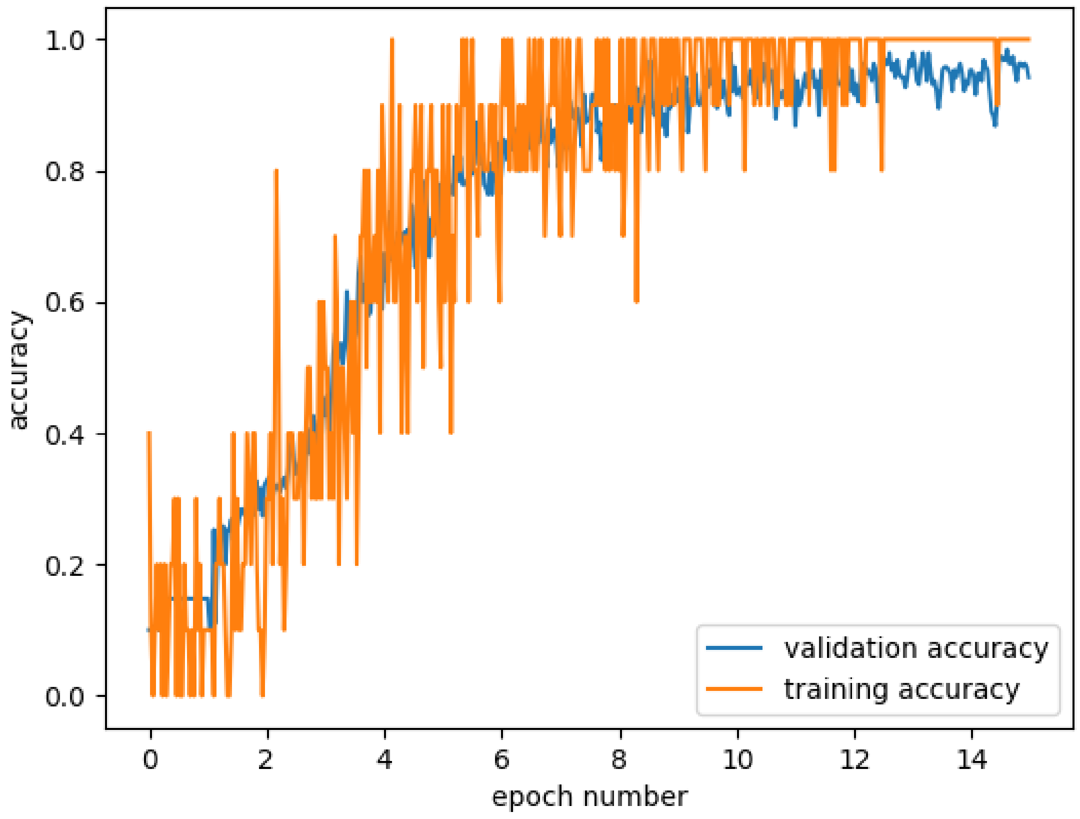


Figure 12: Training and validation accuracy for class prediction over epoches

4.2 Training an validation Loss

The training and validation loss can be shown in figure 13. With decreasing learning rate the fluctuation of training. The cross entropy loss of the training set is decreasing with increasing number of epochs trained and gradually converges.

Still, some over-fitting can be observed after epoch 10 where trianing loss is significantly less than validation loss. Thus, it is advisable to stop training at that time.

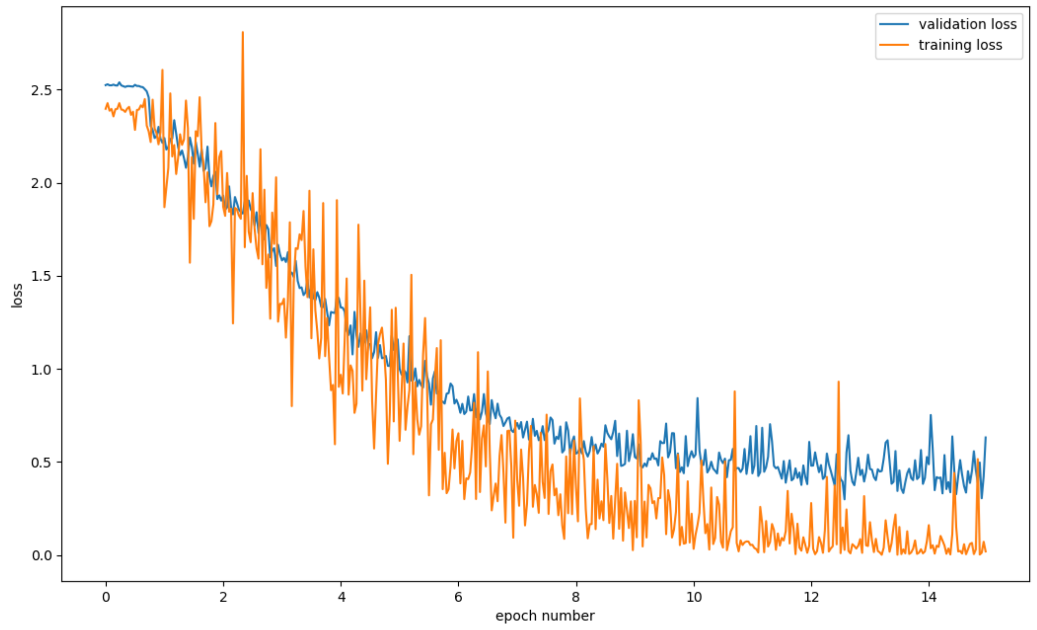


Figure 13: Total training cross-entropy loss as a function of epochs

4.3 Prediction accuracy on test set and confusion matrix

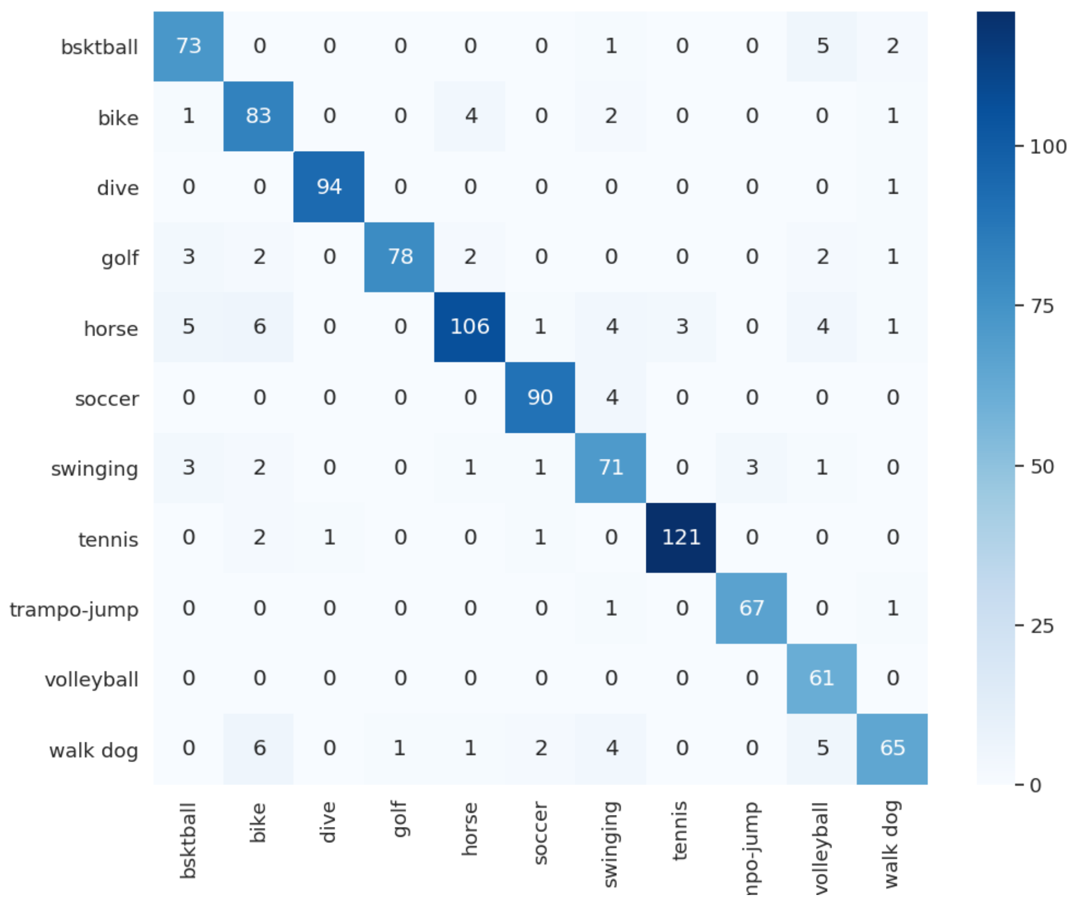
After the training is done, a shuffled testset of 1000 video sequences are tested on the trained C3D small CNN. The confusion matrix of number of correct prediction made for each class and normalized values are shown separately in figure 14 and figure.

Since the testset is shuffled and randomly selected, it is not guaranteed that the class distribution is even in the 1000 test cases. Thus, the normalized confusion matrix better shows the performance of the trained CNN on each class's prediction.

It can be shown that class 'volleyball' has the highest accuracy of 100% on the test set, followed by 'diving' of 99%. The worse performing class is 'a man walking with a dog,' achieving an accuracy of only 77%. This can be explained by the fact that the scene contains two important 'features': 'man' and 'dog'. While 'man' can barely be distinguished fomr other classes, finding a dog in a 64x64 frame seems to be very demanding. 'Diving' and 'Volleyball' are easier for the CNN presumebaly due to that the color of water/volleyball field and the body gesture for both cases help the CNN to extract features more easily.

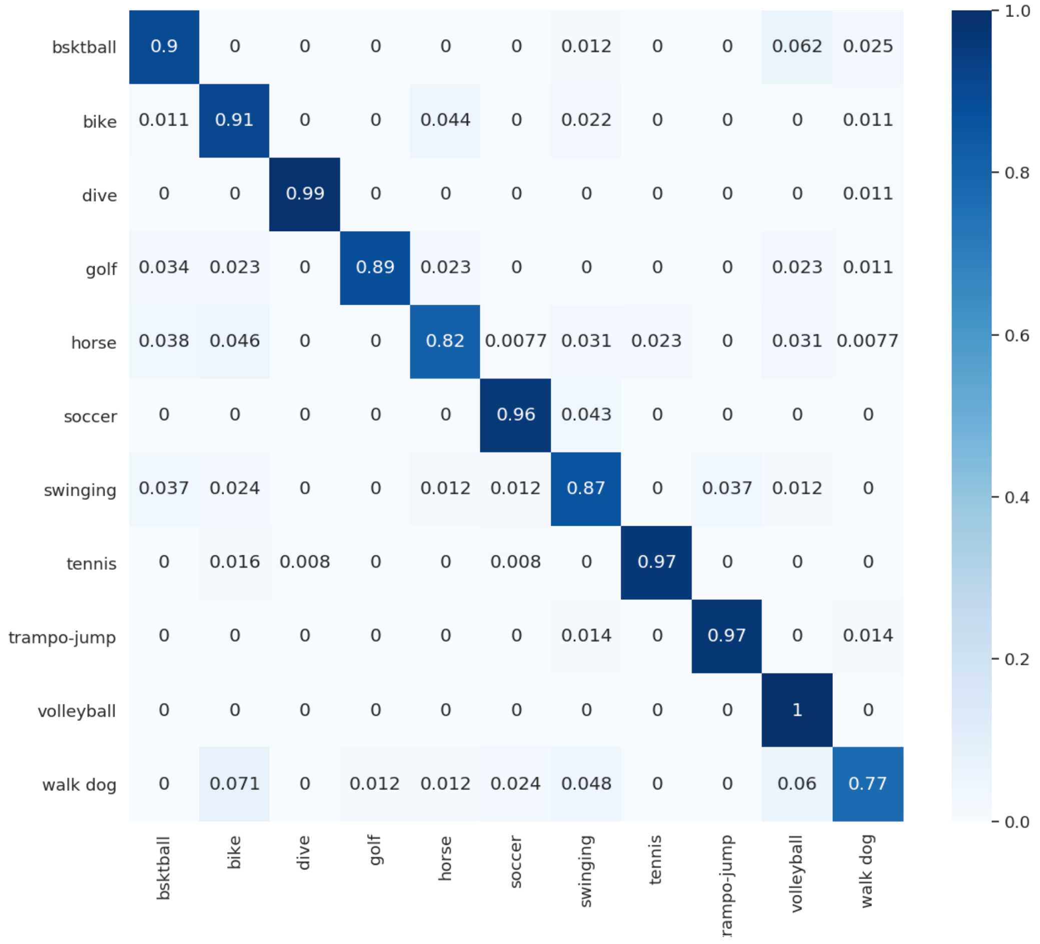
References

[1] https://arxiv.org/abs/1412.0767



	bsktball	bike	dive	golf	horse	soccer	swinging	tennis	npo-jump	volleyball	walk dog
bsktball	73	0	0	0	0	0	1	0	0	5	2
bike	1	83	0	0	4	0	2	0	0	0	1
dive	0	0	94	0	0	0	0	0	0	0	1
golf	3	2	0	78	2	0	0	0	0	2	1
horse	5	6	0	0	106	1	4	3	0	4	1
soccer	0	0	0	0	0	90	4	0	0	0	0
swinging	3	2	0	0	1	1	71	0	3	1	0
tennis	0	2	1	0	0	1	0	121	0	0	0
trampo-jump	0	0	0	0	0	0	1	0	67	0	1
volleyball	0	0	0	0	0	0	0	0	0	61	0
walk dog	0	6	0	1	1	2	4	0	0	5	65

Figure 14: Confusion matrix of prediction results on a test size of 1000



	bsktball	bike	dive	golf	horse	soccer	swinging	tennis	rampo-jump	volleyball	walk dog
bsktball	0.9	0	0	0	0	0	0.012	0	0	0.062	0.025
bike	0.011	0.91	0	0	0.044	0	0.022	0	0	0	0.011
dive	0	0	0.99	0	0	0	0	0	0	0	0.011
golf	0.034	0.023	0	0.89	0.023	0	0	0	0	0.023	0.011
horse	0.038	0.046	0	0	0.82	0.0077	0.031	0.023	0	0.031	0.0077
soccer	0	0	0	0	0	0.96	0.043	0	0	0	0
swinging	0.037	0.024	0	0	0.012	0.012	0.87	0	0.037	0.012	0
tennis	0	0.016	0.008	0	0	0.008	0	0.97	0	0	0
trampo-jump	0	0	0	0	0	0	0.014	0	0.97	0	0.014
volleyball	0	0	0	0	0	0	0	0	0	1	0
walk dog	0	0.071	0	0.012	0.012	0.024	0.048	0	0	0.06	0.77

Figure 15: Normalized confusion matrix