

Report Assignment 5: Convolutional Neural Networks

1. Model Descriptions

1.1. CNN Model (base model on Stanford-40)

The CNN Model was inspired from ResNet20 architecture with some changes. With Residual Networks, the motivation is to use skip connections for getting better accuracy when more layers are added. Therefore, a deep network is more efficient in terms of feature reusability.

Our changed ResNet has the following structure:

1. Input with shape (224, 224, 3)
2. 1 Conv2D layer, with 16 filters
3. 1, 1, 1, 1 residual blocks with 64, 128, 256, and 512 filters
4. AveragePooling2D layer with pool size set to 4
5. Flatten layer
6. Dense layer with 40 output nodes

All kernel sizes are 3x3. Firstly, a helper function is created which takes a tensor as input and adds relu and batch normalization to it. Then a second function for constructing a residual block is created. It takes a tensor x as input and passes it through 2 conv layers. Then adds the input x to y, adds relu and batch normalization, and then returns the resulting tensor. The output of these 2 convolutional layers is y. When parameter downsample is set to true the first convolutional layer uses stride of size 2 to halve the output size and we use a convolutional layer with kernel size 1 on input x to make it the same shape as y. The Add layer requires the input tensors to be of the same shape [1]. Finally, the model architecture is created with a third function that uses the above mentioned functions to generate the ResNet structure. *CNN_model.ipynb* contains the implementation of this model.

1.2. Transfer Learning Model

For the transfer learning model on TV-HI dataset we used the following workflow recommended on keras.io [2]:

1. Take layers from pre-trained CNN model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers.
4. Train the new layers on your dataset.

Specifically, after we load the old model (pre-trained CNN model), we set the weights that have been saved after training the CNN model. Next, we make sure to consider only the layers before the flatten layer. We replace the old flatten and dense layers with new layers: GlobalAveragePooling2D, Batch Normalization, Dense(16) and Dense(4). The latter has output 4 which corresponds to the 4 interaction classes (handshake, high five, kiss and hug). Once the model converged on the new data, we unfreeze all layers while leaving BatchNorm layers frozen. If we do not keep the BatchNorm layers in inference mode, the updates applied to the non-trainable weights will suddenly destroy what the model has learned. This is important because we do not want to update its mean and variance statistics [2]. Next, we retrain the whole model end-to-end with a very low learning rate (1e-4 which

is 1/10 of the original value used on Stanford-40). *TF_TVHI.ipynb* contains the implementation of this model.

1.3. Optical Flow Model

For the optical flow model 3D convolutions were used. Input shape was set to be (16, 224, 224, 1), as we used 16 sequential frames of optical flow where the middle frame (frame 8) corresponded to the middle frame of the video. The optical flow was calculated using the Farnebäck algorithm and then the optical flow was encoded as polar coordinates and stored as jpg images in rgb, that were then encoded as grayscale when loaded. The model consisted of 5 blocks of 3D convolutions, MaxPooling3D, and BatchNormalization with filter sizes 32,32,64,128,128. The number of filters were increased as more complex patterns can be made in later layers, requiring more filters to capture them and combine them. Batch normalisation was used as it speeds up training and makes it easier in general. Two dropout layers with dropout rate 0.2 were included between the second and third block as well as between the fourth and fifth block in order to prevent overfitting. In the chosen model, kernel regularisers L1 and L2 were added to all convolutional layers except the input layer, also to prevent overfitting since the dataset was very small. After the last 3D Convolution, MaxPooling and BatchNormalization block GlobalAveragePoolin3D was called to flatten the convolutions into one vector. After this one dense layer with 512 nodes and a dropout layer was made before the output layer with four nodes corresponding to the four categories. The model loss was calculated with categorical cross entropy and the optimizer was chosen to be Adam as it is a flexible optimiser that adapts to the data. The convolutional layers as well as the second least dense layer used rectified linear activation (ReLU) as it speeds up training and combats the vanishing gradient problem. The output layer used softmax activation to ascribe a probability of each data sample belonging to each category. The model was run for 10 epochs as the amount of data was small and the training showed proneness to overfitting. *OF_model.ipynb* contains the implementation of this model.

1.4. Two-Stream Model

For the Two-Stream (TS) model, the two models trained on optical flow (OF model and Transfer Learning Model on TVHI) were first loaded and their weights frozen. Their final layers were removed and for the OF model the last MaxPooling layer output was reshaped using a Reshape layer to reduce the dimensions from (None, 1, 14, 14, 128) to (None, 14, 14, 128). This was done so that both models had the same spatial extent at the time of concatenation. The two models were concatenated, and an additional 2D convolutional layer was added with filter size 64, followed by batch normalisation and MaxPooling to reduce dimensionality even more. The output from the MaxPooling layer was then flattened and followed by a Dense layer with 256 neurons, batch normalisation and a Dropout layer with dropout rate 0.2. A similar block with Dense layer with 64 neurons, batch normalisation, another dropout layer was then added before the outlayer with 4 neurons and a softmax activation. The convolutional layer after the concatenation was added as the model still needed to be trained and since the concatenated feature maps could provide useful information on the activity when considering both the optical flow and frame data. Two extra dense layers were added in order to find good combinations of the features found in the two input models. All dense and convolutional layers again had rectified linear activation except for the last. The model was set to be trained for 10 epochs. The architecture of the final layers of the TS model can be seen in Figure 1. *TS_model.ipynb* contains the implementation of this model.

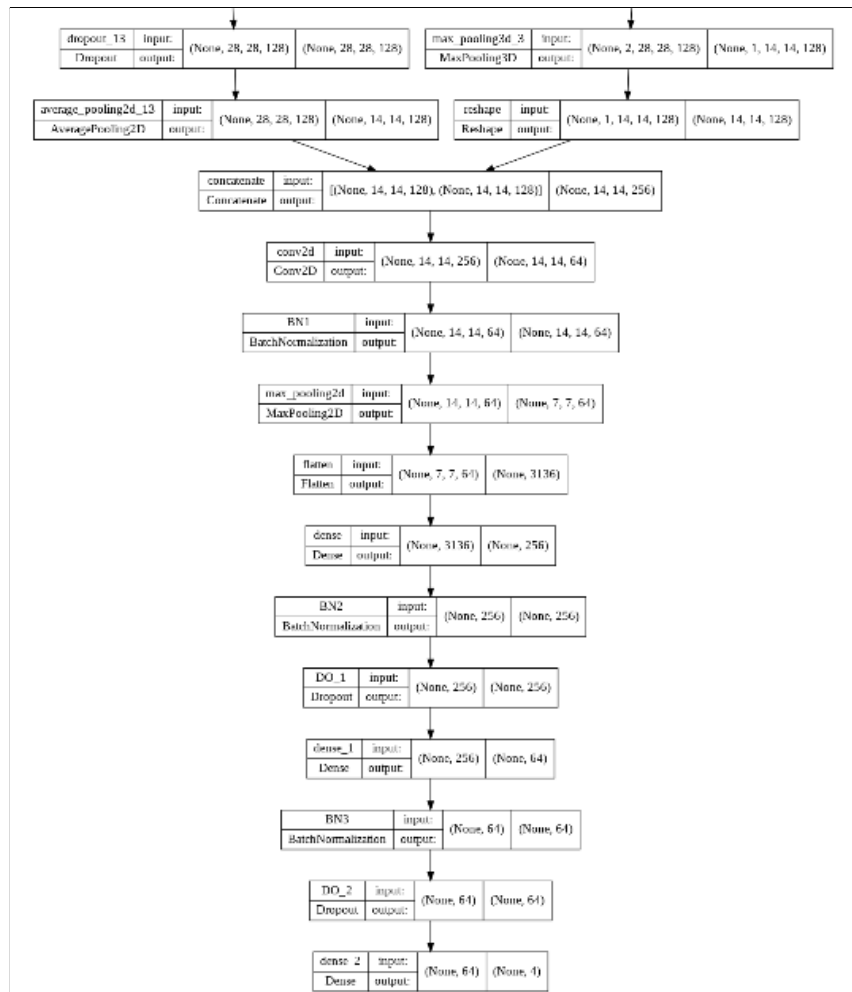


Figure 1. The architecture of the fusion in the TS model

2. Results

Dataset	Model	Top-1 acc. Train	Top-1 loss Train	Top-1 acc. Test	Top-1 loss Test	Model size (GB)
Stanford40	Frames	0.3416	2.3509	0.2301	3.0569	1.692
TV-HI	TL	0.3764	1.2925	0.2400	1.3858	1.685
TV-HI	Optical flow	0.9522	1.8065	0.3333	4.4150	2.401
TV-HI	Two-stream	////////	////////	////////	////////	3.337

Table 1. Overall Results

2.1. CNN Model

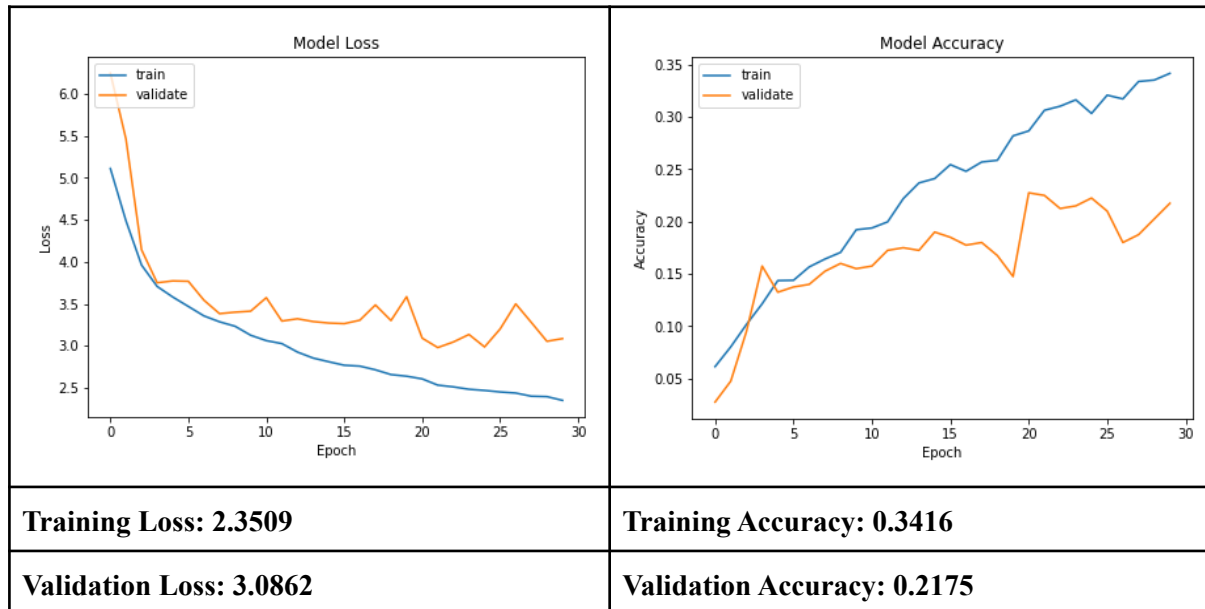


Table 2. CNN Model loss/accuracy

As observed in the plot, the validation loss gets higher in spikes around epoch 9, 16 and 26, while the training loss continues to lower. This might indicate that the model overfits in certain places and more tweaks can be done to the model. However, the accuracy when evaluating the test set (as shown in Table 0) is above the expectations of 10-20%, reaching 23% (0.2301) with data augmentation included.

2.2. Transfer Learning Model

Given an overall observation of the graphs, the validation loss is decreasing in orders of 0.001, which means that not enough/representative validation data is provided and the validation graph looks like a straight line. By fine-tuning and changing the learning rate to 0.0001 we could observe the increase in training accuracy, but again it seems like more validation images are required to increase validation accuracy. When evaluated on the test dataset, the accuracy is however lower at around 24% which indicates a random baseline. Finally, since this model also includes data augmentation the graphs show less overfitting compared to the model without data augmentation (see graph at choice tasks section).

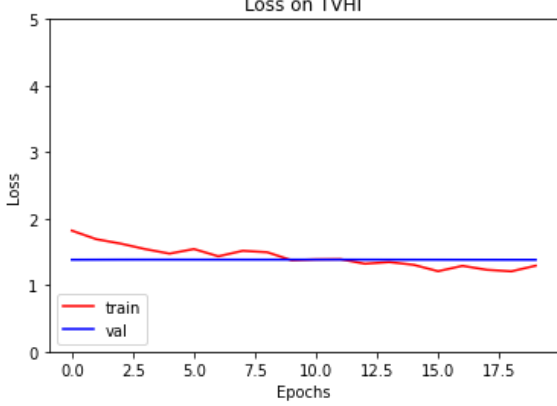
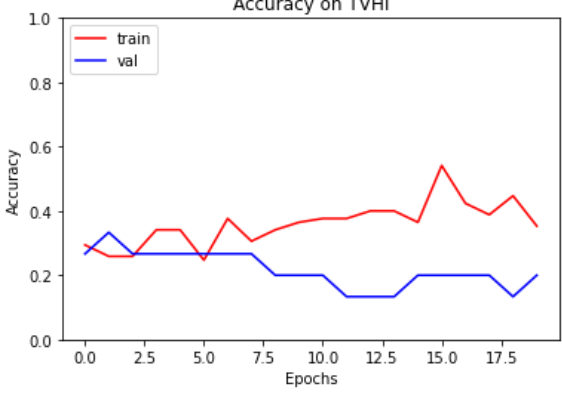
	
Training Loss: 1.2925	Training Accuracy: 0.3764
Validation Loss: 1.3830	Validation Accuracy: 0.3333

Table 3. Transfer Learning Model loss/accuracy

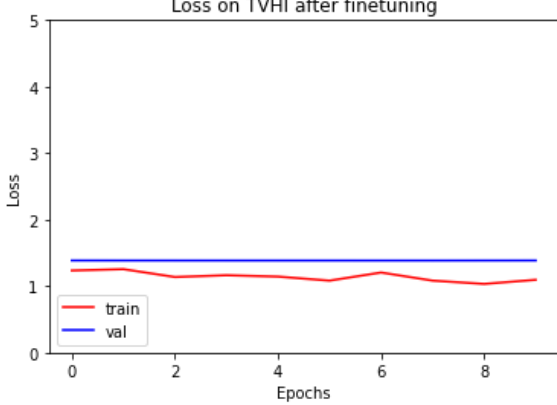
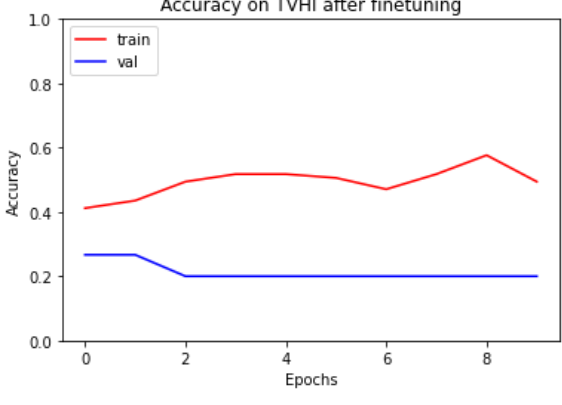
	
Training Loss: 1.0946	Training Accuracy: 0.49411
Validation Loss: 1.3856	Validation Accuracy: 0.2000

Table 4. Transfer Learning Model after fine tuning loss/accuracy

2.3. Optical Flow Model

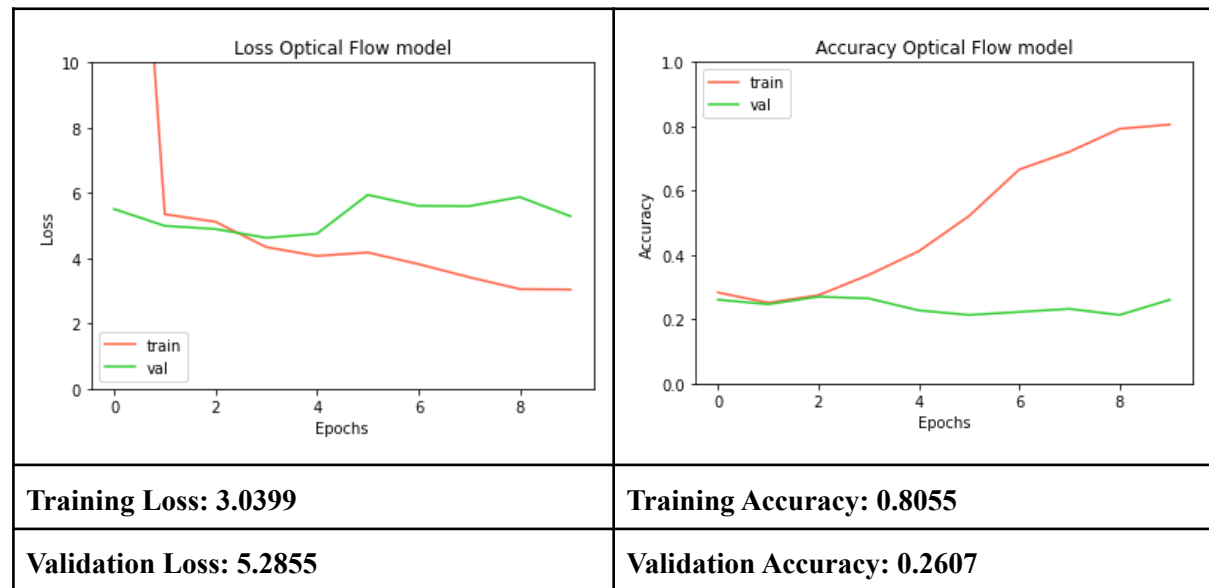


Table 5. Filter Model loss/accuracy

The training of the optical flow model happened with a split dataset of 85 training videos and 15 validation videos, each sample consisting of a stack of 16 frames. The training took place over 10 epochs, where the training accuracy increased steadily while the validation accuracy fluctuated between 0.15 to 0.32 during the entire training time, showing an accuracy equal to chance. This was likely due to the validation set being too small or not having representative examples. When inspecting part of the validation data it could also be seen that some samples were deprecated, probably due to bad encoding in the original videos, meaning that the optical flow could not be obtained. In the future, the problem of the small dataset could possibly be ameliorated by using cross validation or data augmentation for the optical flow dataset as well.

2.4. Two-Stream Model

Due to complications during the work and lack of time, we did not manage to run the model. When calling the fit function on the model, the error in figure 2 was observed. We suspect that the problem might have to do with the data generator function that fits two data generators to the TS model, where one generator feeds 16 frames at once and the other feeds 1 frame at a time. However, further investigation might have yielded a different conclusion.

```
Node: 'model_1/tf.nn.relu_126/Relu'
input depth must be evenly divisible by filter depth: 224 vs 3
[[{{node model_1/tf.nn.relu_126/Relu}}]] [Op: __inference_train_function_28543]
```

Figure 2. The error message when calling TS_model.fit() function.

3. Task Selection

3.1. Data augmentation for Stanford-40

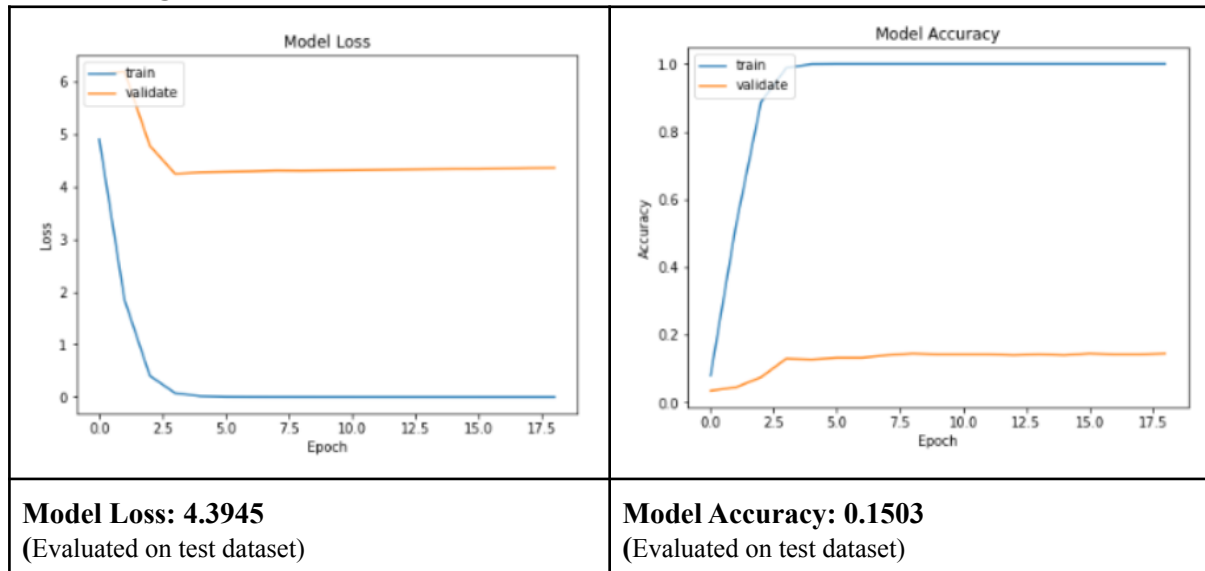


Table 6. CNN Model loss/accuracy without data augmentation.

The main reason for applying data augmentation is the graph above. At the first run of the model we could observe that overfitting occurs and data is not complex enough. Hence, we perform augmentation through layers such as normalization, random rotation, translation, horizontal flip and random zoom. These layers apply random augmentation transforms to a batch of images. They are only active during training. Their factor is chosen based on the Loss/Accuracy graph. The outcome for applying Data Augmentation can be seen in the Results section above.

3.2. Data augmentation for TV-HI

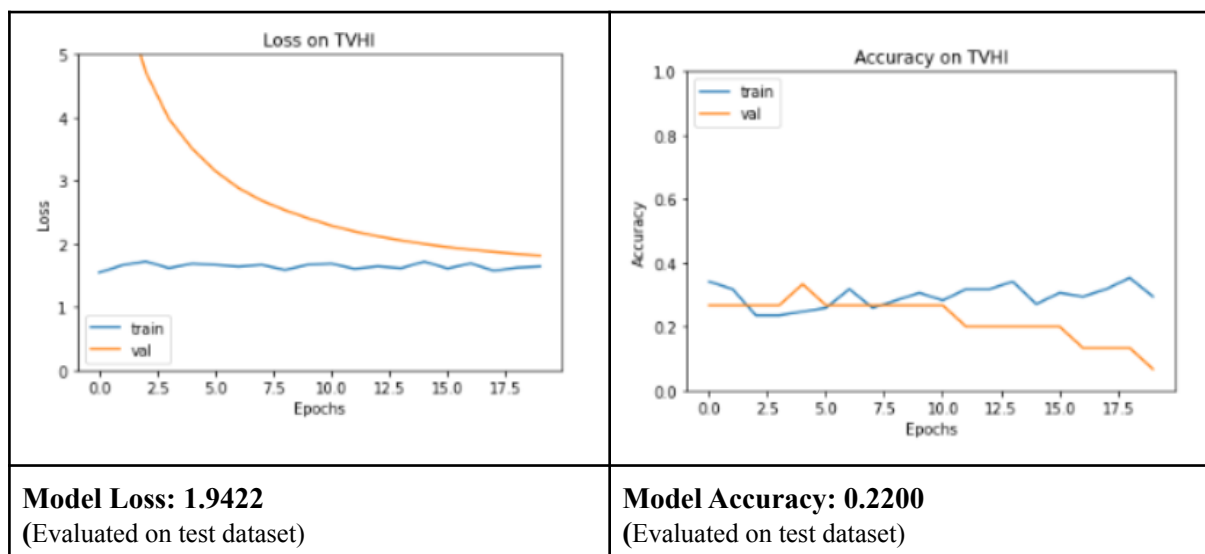


Table 7. Transfer TV-HI Model loss/accuracy without data augmentation.

We apply data augmentation for TV-HI due to the lack of data. Especially for this dataset, we apply a RandomContrast with a factor of 0.7 after observing important differences in brightness throughout the images. Unfortunately, because the validation set is too small or does not have representative examples, we could not make accurate evaluations on the test dataset but we suppose that adjusting the contrast during training would greatly improve the accuracy. The outcome for applying Data Augmentation can be seen in the Results section above at the Transfer Learning Model.

3.3. Use a kernel regularizer for your convolutions

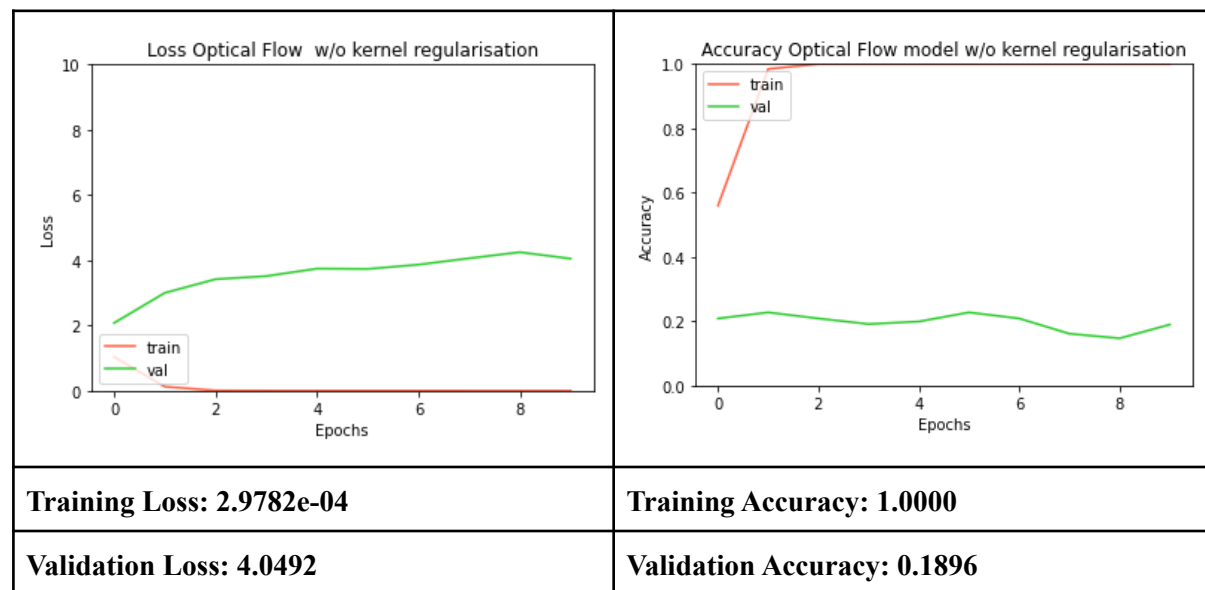


Table 8. Optical flow model without kernel regularisation

For the Optical Flow model the original model presented before was run with kernel regularizers L1 and L2 on all convolutional layers but the first in the model. As documented before, the validation accuracy was still quite low on the TVHI-optical flow dataset, with the model overfitting quite largely on the dataset. However, without regularisation, the overfitting is even more severe as can be seen in the plots above. The overfitting happens in earlier epochs compared to with the Optical Flow model with regularisation, and the training loss is also more severe. This has also led to a smaller validation accuracy as the model does not generalise well to new data.

3.4. Stack a fixed number (e.g., 16) of optical flow frames together (max 15 points)

In order to fit a stack of frames to the models, a data generator by following a tutorial[3]. After the optical flow had been calculated the optical flow images were stored in a folder for each video. Depending on dataset (training, validation, or test) these videos were then sorted into a specific folder and each frame were encoded as a csv file containing the path to the video using a Pandas dataframe. These were then used to create a data list that for each video contained csv files for each frame in the video as well as the class it belonged to. These lists were then upon run time passed to a custom made data generator using the Keras Functional API that created a data object of a determined batch size that yielded the next sample at each time step during training.

For the TS model, a custom function named joinedGens (found in utils.ipynb) was made that could fit two data generators at once and yielded the next samples during run time.

Link to model weights:

<https://drive.google.com/drive/folders/1igqqY1TecDVdvoARS-dc1ALi5jFaLZKp?usp=sharing>

Bibliography

- [1]<https://towardsdatascience.com/building-a-resnet-in-keras-e8f1322a49ba>
- [2]https://keras.io/guides/transfer_learning
- [3]https://medium.com/@anuj_shah/creating-custom-data-generator-for-training-deep-learning-models-part-3-c239297cd5d6