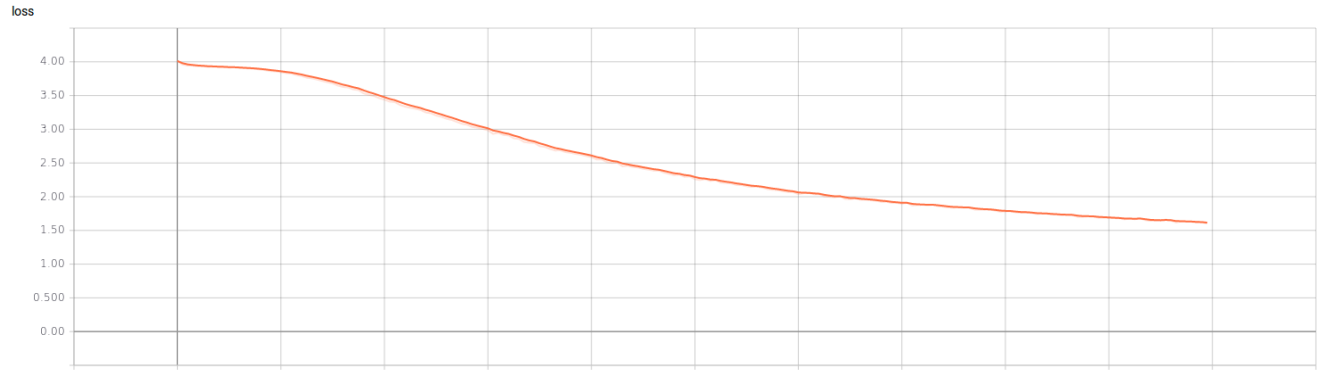
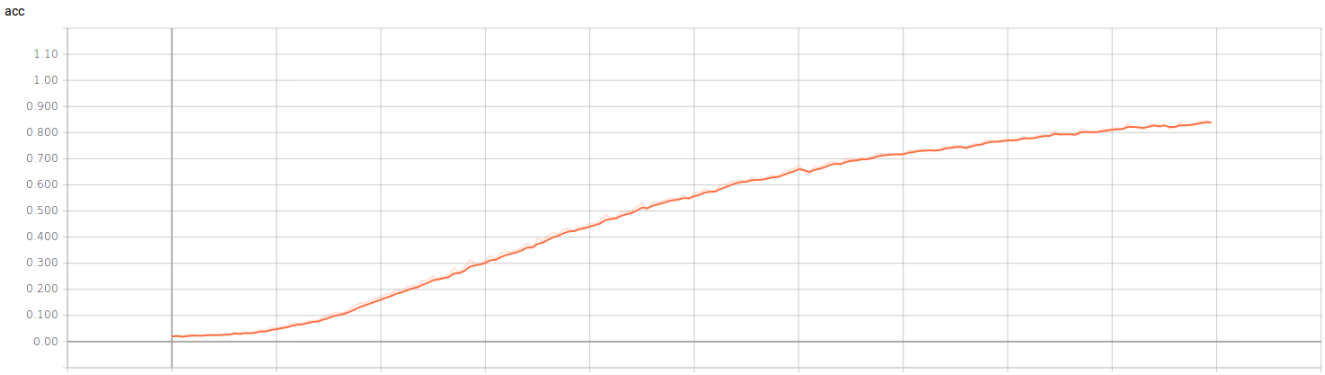
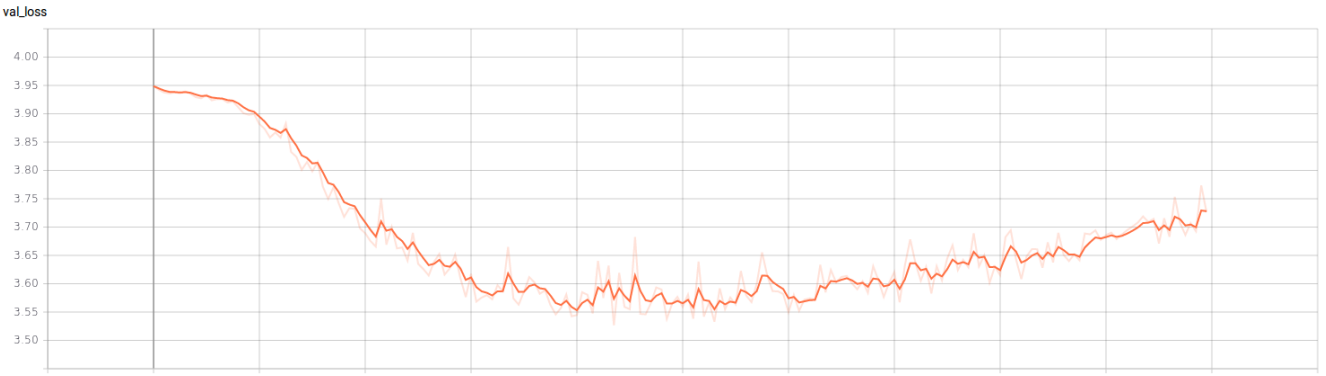
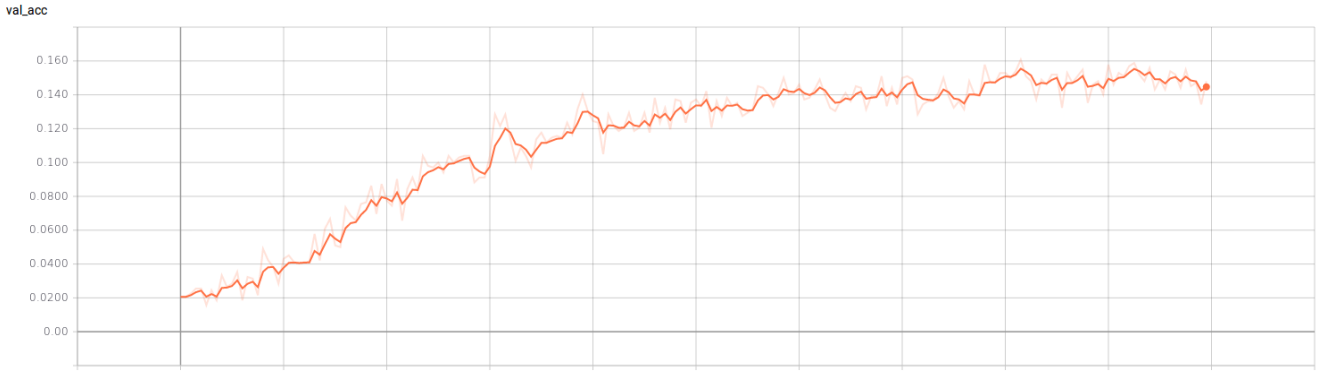
I use Tensorflow-Keras to implement this assignment.

Q1.1
Here I use a three-layer MLP to do the classification. Note that the input shape of the network is (batch_size=10, 10, 512). In the model, the input is first reshaped to a 2-d tensor of (100, 512) to fit into the dense layer. Then I have two hidden layers with 512 units, followed by relu and 0.5 dropout. Last the output is reshaped back to 3-d tensor and averaged along the second axis, to get the mean score for each sequence.
Cross entropy loss and Adam optimizer is used here. Learning rate is 0.0001. ¼ of train set is split to be validation. Using a batch size of 10 and training for 200 epochs, here is the result.





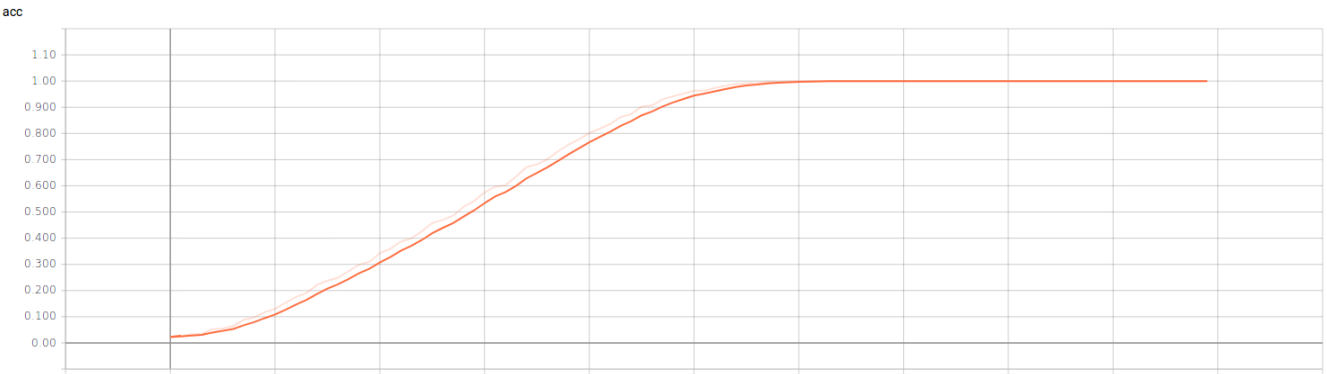


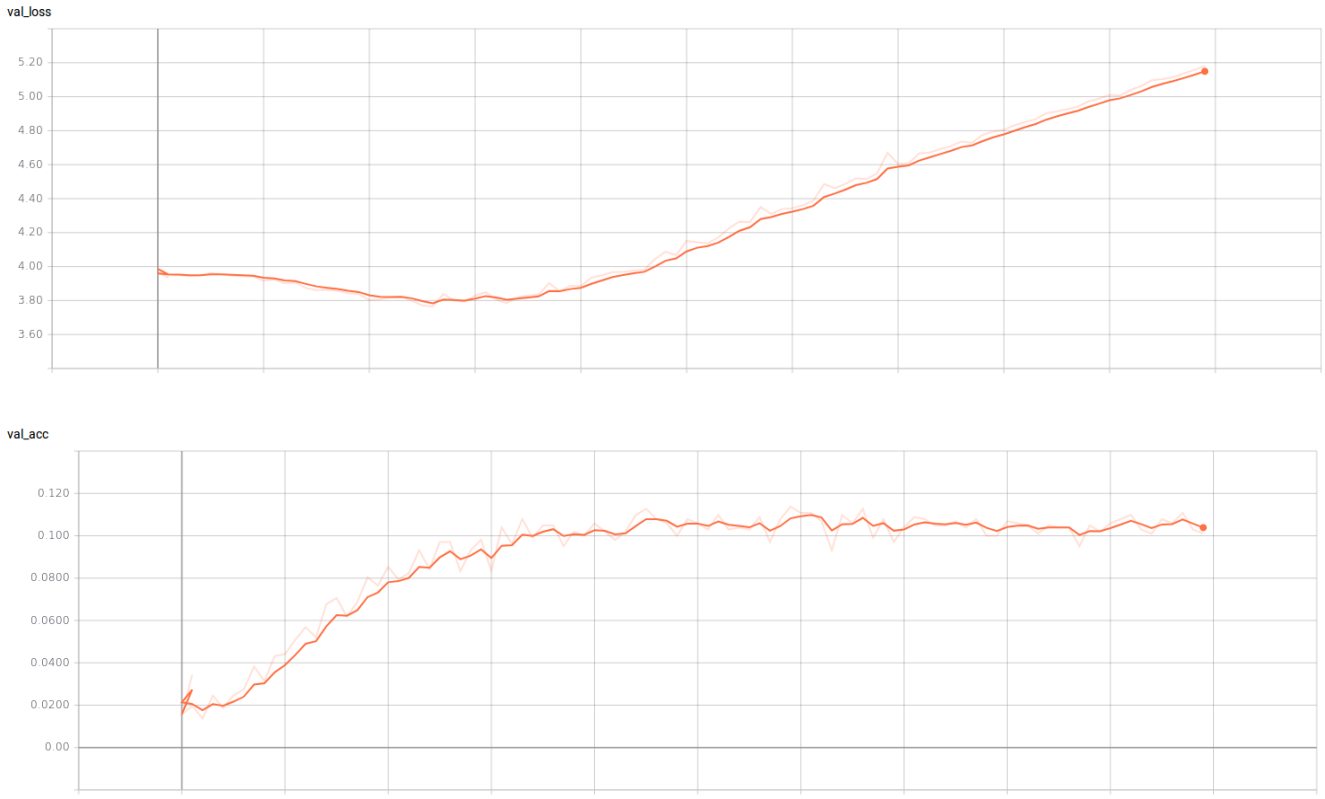


We can see that the model is converging fast on train set, and although it is obviously overfitting, validation accuracy can still achieve around 15%. I do not train more epochs because I see the val loss already increasing and worry that the val result might drop.

Q1.2

Here I use a LSTM with (10,512) input, followed by a 256-hidden-unit fc layer, and then the softmax cross-entropy loss. Adam optimizer is used here. Learning rate is 0.0001. Using a batch size of 32 and training for 100 epochs, here is the result.





The model is overfitting badly, with training loss goes to 0 and training accuracy 100%. The validation accuracy is barely above 10%

Q2

1. Because regression loss always yields continuous results, which is not good in surface normal estimation tasks and can cause blurry result. Instead a classification loss with discretized value can do the job.

2. Batch normalization is used to normalize a batch input of a neural net. It will calculate the mean and variance of the batch, and make updates to global mean and variance using a decay, and normalize the batch data with global mean and variance. It is useful when you have vanishing gradient problem with deep network. It can normalize so that the data and gradient do not vanish. Difference between training and testing phases is that during training you update the mean and variance with current batch mean and variance, while during testing you just use the population rather than mini-batch statistics.

3. It depends on whether these two streams are trained for same or similar tasks. For example, in the two-stream CNN for action recognition, one stream is spatial stream and the other is temporal stream, which are totally different. And in the Siamese and Triplet network, streams share weights because we want the same embeddings.

4. Initialize with ImageNet-pretrained weights. To avoid overfitting, we can use more data, add dropout layers, stop training the net when validation error increases for several iterations.

5. Discretize the continuous variable.

6. Cross-entropy loss.

7. Because first of all CNN deals with 2-d data, like images, while other tasks like NLP or RL usually have 1-d data. And also, CNN is related to how vision works (receptive fields), and some filter-based traditional CV method.

8. Based on my prior experience and some search. For example, momentum is usually set to 0.9 for most task. And Adam optimizer is always a good option. For learning rate, I will start with 0.01 and see the loss curve. If it is not decreasing I might try smaller ones, and if it is decreasing slowly I will try bigger ones.