Report on project submission 3 for CSCE 636- Spring 2021

Bahareh Alizadeh Kharazi

March 11, 2021

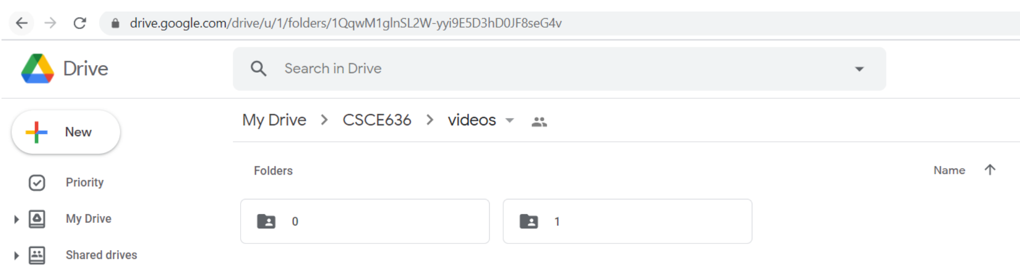
Abstract

In this project, the goal is to detect the action of shaking head in video clips. A transfer learning approach is proposed in this project which was developed based on using ResNet50 and replacing its first layer with one LSTM layer and three other dens layers. A dataset is created by the author which 50% of it is labeled as shaking head. The test accuracy of this model on the 30% of the entire dataset was achieved as 77% which is acceptable. However, the limited amount of videoclips in the dataset, still questions the generalizability of the model.

Collecting training dataset

The topic selected for this project was "Shaking-head". To create a dataset for training the model, I had to search for videoclips that have the action "shaking-head" in one clip. Since most videos available on the web have different actions in one clip, I decided to find GIF files (The Graphics Interchange Format) that are commonly used file types in social media that focuses on one action at a time. This trick helped me to train my model easily on the dataset with limited amount of data for my first submission.

For the first submission, I collected 50 photos containing shaking-head action and 50 photos containing other actions. To organize the data I named all vidoes with shaking-head action and videos with non-shaking-head action as "shakinghead_number" and "other_number", respectively. All these folders were moved to Google Drive folders to be used in Google Collab.



Folder 0 contains all videos with other actions and folder 1 contains videos with shaking-head action.

Preprocessing of the dataset

To prepare videoclips for my model, first I load the modules that I need. I also, mount the Google Drive in the Google Collab:

```
[ ] %tensorflow_version 1.x

    TensorFlow 1.x selected.
```

```
[ ] pip install pytictoc

    Requirement already satisfied: pytictoc in /usr/local/lib/python3.7/dist-packages (1.5.1)
```

```
[ ] import numpy as np
    import keras
    import sklearn as sk
    import cv2
    from keras.applications.resnet import ResNet50
    from keras.utils.vis_utils import plot_model
    from keras.models import Model
    from keras.layers import Input, Dense, Flatten
    import pytictoc
    import math
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    import glob
    import os
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

    Mounted at /content/drive
```

Then videos in each folder will be converted to frames (as image) and saved to other folder. The process for videos with label 1 is shown below:

```
time_print = pytictoc.TicToc()
time_print.tic()
videos = []
labels_2d = []
num_frames = 100
frame_rate = 5
time_3d = []
time_2d = []
############################## Videos with label = 1
i=0
for path in glob.glob('/content/drive/My Drive/CSCE636/videos/1/*.mp4'):

  vidcap = cv2.VideoCapture(path)
  fps = vidcap.get(cv2.CAP_PROP_FPS)
  success, image = vidcap.read()
  frames = []


  time = []
  count = 0  # control to have the same number of frames
  count_fps = 0
  while success:


    success, image = vidcap.read()
    count += 1
    if(type(image).__module__ == np.__name__):
      new_image = cv2.resize(image, (224,224), interpolation = cv2.INTER_AREA)
```

```
        frames.append(new_image)
        time.append(count_fps*(1/fps))
        count_fps += 1
        if count==num_frames:
          count_fps = 0
          print("Frames_", str(count),", video_", str(i), "Done, Time_Elapsed: ", str(round(time_print.tocvalue(),3)), " Seconds")
          videos.append(frames)
          time_3d.append(time)
          count = 0
          frames = []
          time = []

    if (count < num_frames):
      while (count > 0 and count <= num_frames):
        frames.append(new_image)    # if the number of frames is lower than the num_frames, repeat the last image to reach num_frames
        count +=1
        time.append(count_fps*(1/fps))
        count_fps += 1

      videos.append(frames)
      time_3d.append(time)

    i+=1
    print("Video_", str(i),"Done, Time_Elapsed: ", str(round(time_print.tocvalue(),3)), " Seconds")


videos_2d_len = len(videos)
```

Then all images will be reshaped to a single shape size.

```
#Correcting shapes of images
ind = list(np.random.randint(0,len(videos_3d)-1,size=len(videos_3d)))
videos_3d_temp = videos_3d[ind]
videos_2d_shuffled = np.reshape(videos_3d_temp, (-1,224,224,3))
videos_2d_shuffled = videos_2d_shuffled[:len(videos_2d_shuffled):frame_rate]

time_3d_temp = time_3d[ind]
time_2d_shuffled = np.reshape(time_3d_temp, (-1,1))
time_2d_shuffled = time_2d_shuffled[:len(time_2d_shuffled):frame_rate]
time_2d_shuffled = time_2d_shuffled.T[0]

labels_2d_temp = labels_2d[ind]
labels_2d_shuffled = labels_2d_temp.flatten()
labels_2d_shuffled = labels_2d_shuffled[:len(labels_2d_shuffled):frame_rate]
```

Since processing all images every time is time consuming, I saved all the frames as numpy file to speed up the process for further work. Although currently the number of training image in our dataset is limited, but this approach will save time when my dataset gets larger later.

Selecting a model

I used transfer learning on ResNet50 by excluding the first layer and adding four other layers to it (one ConvLSTM2D and three dense layers).

First we load the modules:

```
import numpy as np
import keras
import sklearn as sk
import cv2
from keras.applications.resnet import ResNet50
from keras.utils.vis_utils import plot_model
from keras.models import Model
from keras.layers import Input, Dense, Flatten, LSTM, ConvLSTM2D, Reshape, Conv2D, Dropout, BatchNormalization
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.layers.wrappers import TimeDistributed
import glob
import os
import tensorflow as tf
import json
from keras.models import model_from_json
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Then we load the numpy data that we created in the previous section:

```
#Load data with npy formats
X = np.load("/content/drive/My Drive/CSCE636/Dataset/raw_data.npy")
y = np.load("/content/drive/My Drive/CSCE636/Dataset/raw_label.npy")
times = np.load("/content/drive/My Drive/CSCE636/Dataset/times.npy")
```

Each second in the videoclips is going to be divided to 20 frames:

```
#Generating 20 frames out of each seconds of videos
num_frames = 20
len_X = len(X)
num_videos = len_X/num_frames
```

Also, we took 70% of the dataset as the training set and the rest for the test set.

```
#70% of data splitted for training 30% for testing
percentage = 0.7
#splitting the data
X_train = X[:int(percentage*num_videos)*num_frames]
y_train = y[:int(percentage*num_videos)*num_frames]
X_test = X[int(percentage*num_videos)*num_frames:]
y_test = y[int(percentage*num_videos)*num_frames:]
times_train = times[:int(percentage*num_videos)*num_frames]
times_test = times[int(percentage*num_videos)*num_frames:]
```

Training and testing the model on the dataset

Then we apply transfer learning as stated before, without training the wights of 49 layers of the ResNet model. We only train the weights for the added four layers.

```
[ ] #applying transfer learnign (the first layer is removed)
    ResNet = ResNet50(include_top=False)

    ResNet50_Model_base = Model(inputs = ResNet.input, outputs = ResNet.get_layer("conv5_block3_add").output)

    all_layers = ResNet50_Model_base.layers

    for layer in ResNet50_Model_base.layers:
        layer.trainable = False #ResNets weights will not be trained

    #Assuming the input frames have the Height: 224 and Width: 224 pixels
    Input_to_ResNet = Input(shape=(224, 224, 3),name = 'ResNet50')

    ResNet_last_layer = ResNet50_Model_base(Input_to_ResNet)
```

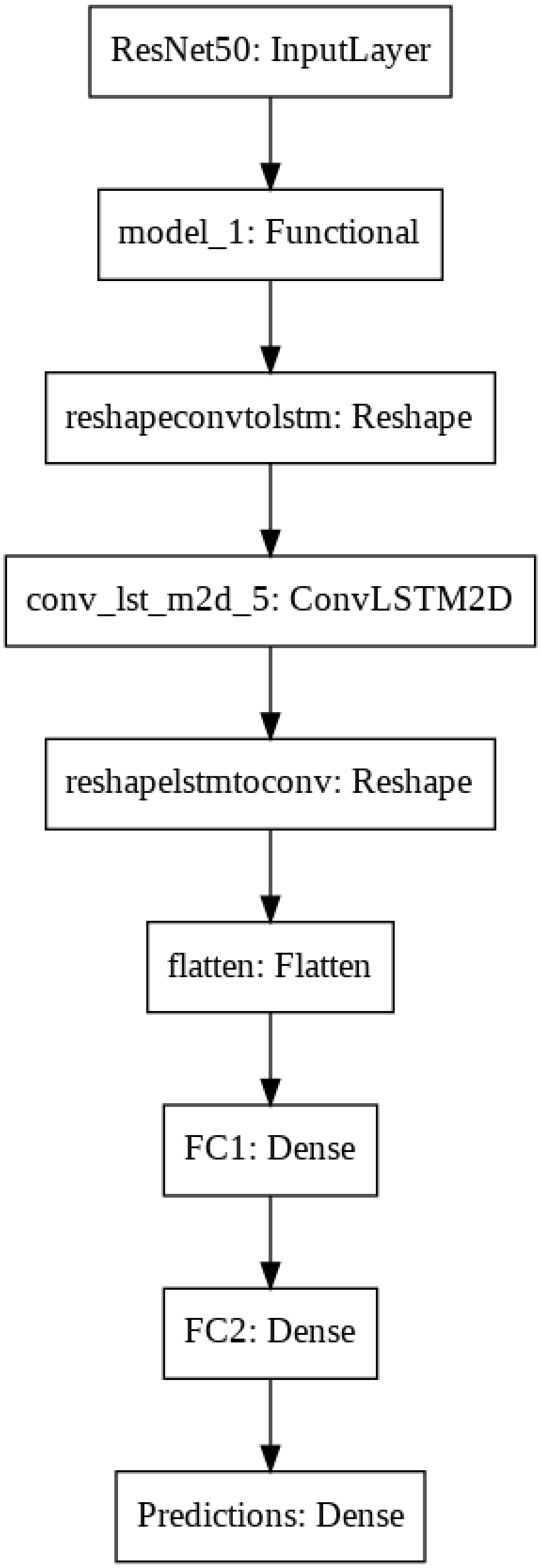
We check the name of the last layer in ResNet (below) and add it as the output in the code above:

```
[ ] #showing the last layer in ResNet
    ResNet.output_names

    ['conv5_block3_out']
```

Here, four layers are added to the ResNet model. The first one is ConvLSTM2D and the other three are three dens layers (two with relu active function and one with sigmoid active function). The last layer has one node because our output is binary (0 or 1).

```
x = ResNet_last_layer #Modified ResNet

#Add the fully-connected layers
conv_to_LSTM_dims = (1,7,7,2048)
x = Reshape(target_shape=conv_to_LSTM_dims, name='reshapeconvtolstm')(x)
x = ConvLSTM2D(filters=8, kernel_size=(3, 3), input_shape=(None, 20, 7, 7, 2048), padding='same')(x)  #input_shape=(samples, time, rows, cols, channels)

LSTM_to_conv_dims = (7,7,8)
x = Reshape(target_shape=LSTM_to_conv_dims, name='reshapelstmtoconv')(x)

x = Flatten(name='flatten')(x)
x = Dense(512, activation='relu', name='FC1')(x)
x = Dense(1024, activation='relu', name='FC2')(x)
x = Dense(1, activation='sigmoid', name='Predictions')(x) #1 for binary output
ResNet_for_TL = Model(Input_to_ResNet, x)
ResNet_for_TL.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

plot_model(ResNet_for_TL)
```

The plot of the model is shown below:


ResNet50: InputLayer
model_1: Functional
reshapeconvtolstm: Reshape
conv_lst_m2d_5: ConvLSTM2D
reshapelstmtoconv: Reshape
flatten: Flatten
FC1: Dense
FC2: Dense
Predictions: Dense

Now we fit the ResNet model on our dataset with batch_size=20 and epoch=20. The training accuracy is achieved 99.75% and the test accuracy is obtained 82.64%. Because we have limited amount of data for the first submission, the accuracy varies a lot. By adding more data, our model will be more robust in terms of its accuracy.

```
ResNet_for_TL.fit(x=X_train, y = y_train, batch_size=20, epochs=20, validation_data=(X_test, y_test))
```

```
Epoch 1/20
81/81 [==============================] - 8s 74ms/step - loss: 0.1359 -
accuracy: 0.9370 - val_loss: 0.6610 - val_accuracy: 0.8681
Epoch 2/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0189 -
accuracy: 0.9938 - val_loss: 0.8774 - val_accuracy: 0.8403
Epoch 3/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0245 -
accuracy: 0.9895 - val_loss: 1.4699 - val_accuracy: 0.7264
Epoch 4/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0369 -
accuracy: 0.9883 - val_loss: 0.8230 - val_accuracy: 0.8153
Epoch 5/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0167 -
accuracy: 0.9938 - val_loss: 0.9994 - val_accuracy: 0.8611
Epoch 6/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0408 -
accuracy: 0.9883 - val_loss: 1.0026 - val_accuracy: 0.8389
Epoch 7/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0230 -
accuracy: 0.9901 - val_loss: 0.9920 - val_accuracy: 0.8306
Epoch 8/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0157 -
accuracy: 0.9957 - val_loss: 1.0627 - val_accuracy: 0.8347
Epoch 9/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0093 -
accuracy: 0.9975 - val_loss: 1.2372 - val_accuracy: 0.8361
Epoch 10/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0067 -
accuracy: 0.9975 - val_loss: 1.4526 - val_accuracy: 0.8347
Epoch 11/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0105 -
accuracy: 0.9975 - val_loss: 1.0312 - val_accuracy: 0.8236
Epoch 12/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0086 -
accuracy: 0.9975 - val_loss: 1.4540 - val_accuracy: 0.8417
Epoch 13/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0099 -
accuracy: 0.9951 - val_loss: 2.5456 - val_accuracy: 0.7361
Epoch 14/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0678 -
accuracy: 0.9846 - val_loss: 0.8886 - val_accuracy: 0.8069
Epoch 15/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0336 -
accuracy: 0.9895 - val_loss: 1.1932 - val_accuracy: 0.8417
Epoch 16/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0709 -
accuracy: 0.9753 - val_loss: 0.7298 - val_accuracy: 0.8153
```

```
Epoch 17/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0135 -
accuracy: 0.9957 - val_loss: 0.6954 - val_accuracy: 0.8500
Epoch 18/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0096 -
accuracy: 0.9969 - val_loss: 0.9194 - val_accuracy: 0.8389
Epoch 19/20
81/81 [==============================] - 5s 59ms/step - loss: 0.0102 -
accuracy: 0.9963 - val_loss: 1.2448 - val_accuracy: 0.8208
Epoch 20/20
81/81 [==============================] - 5s 60ms/step - loss: 0.0091 -
accuracy: 0.9975 - val_loss: 0.8940 - val_accuracy: 0.8264
<tensorflow.python.keras.callbacks.History at 0x7f33ebea6390>
```

The plot of the ResNet model's base is shown below (you need to zoom in to the picture):

Testing the model on video clips on youtube

Now that we have our model trained on our dataset, we can test our model on new videoclips. The process of testing the model is elaborated in the following.

We download YouTube videoclips and move them into the folder in the Google Drive and set the test path to its directory:

```
path_test = "/content/drive/My Drive/CSCE636/Youtube_test/test2.mp4"
```

Again, we convert the video into frames and save them as image files. We randomly select frames out of each video.

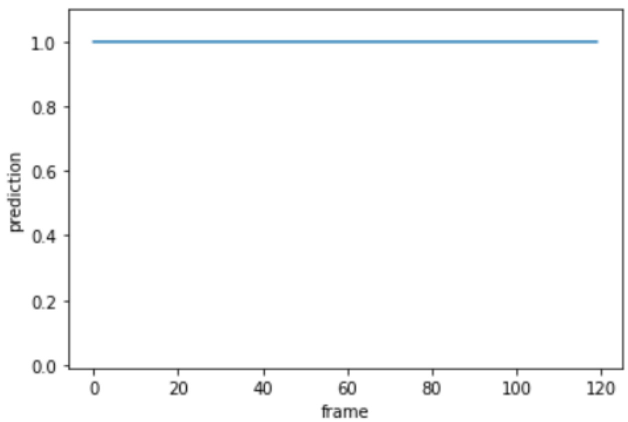
Here we call the model to predict the label of frames in the test data:

```
[ ]  u_test = ResNet50_loaded.predict(videos_2d_shuffled)
```

And then we plot the predicted label for each frame in the video. For this sample video, the shaking-head action is detected which was correct. For each video lips in the test set, we repeat the same process and then we upload true positive samples in the ilab website.

```
plt.plot(u_test)
plt.ylim(-.01,1.1)
plt.ylabel('prediction')
plt.xlabel('frame number')
```

```
Text(0.5, 0, 'frame')
```



Project submission 3 (Date: 03/11/2021)

Collecting training dataset

In the third submission of the project I changed my topic to Cooking/Cutting. I collected 18 long video files (total size was 57.3 MB) that contained cutting activity. Some video clips were collected from a publicly available dataset and other videos were collected from YouTube.

Preprocessing of the dataset

The preprocessing section was not changed comparing to the previous submission. All videos were uploaded to the Google Colab frames were extracted from videos and saved as numpy files.

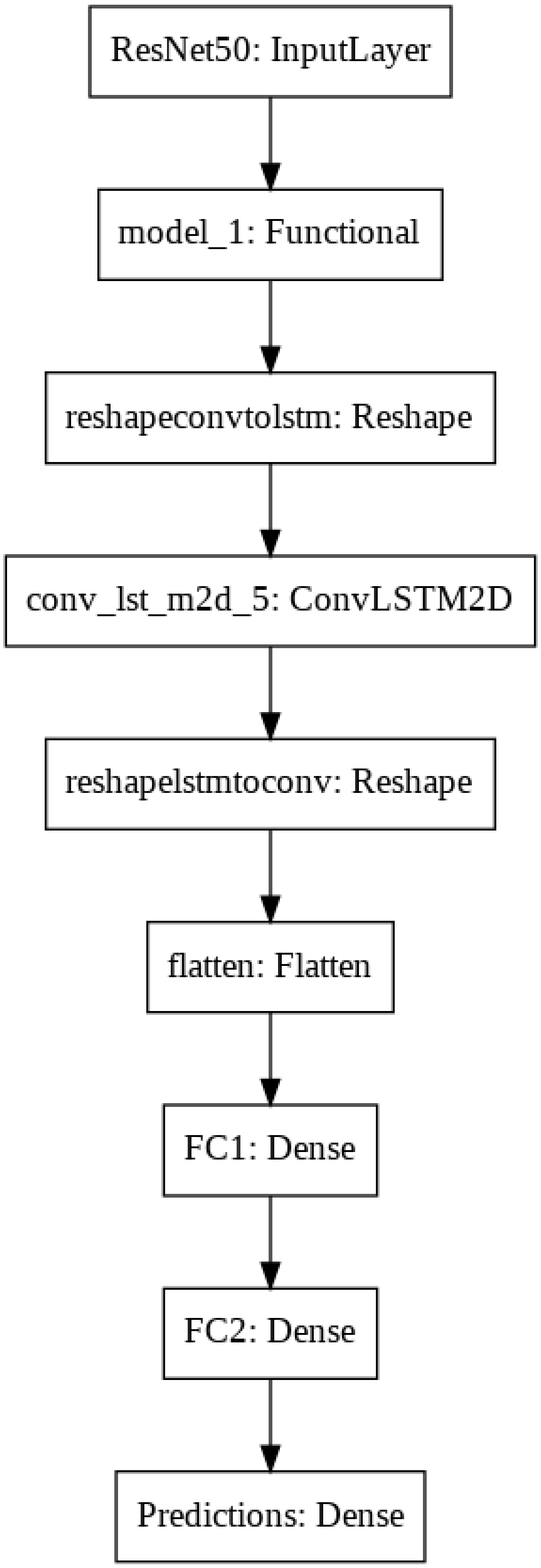
Selecting a model

A transfer learning is used on ResNet50 by excluding the first layer and adding four other layers to it (one ConvLSTM2D and three dense layers).

Each second in the videoclips is going to be divided to 20 frames:

Also, we took 70% of the dataset as the training set and the rest for the test set.Then we apply transfer learning as stated before, without training the wights of 49 layers of the ResNet model. We only train the weights for the added four layers.

Here, four layers are added to the ResNet model. The first one is ConvLSTM2D and the other three are three dens layers (two with relu active function and one with sigmoid active function). The last layer has one node because our output is binary (0 or 1).


ResNet50: InputLayer
model_1: Functional
reshapeconvtolstm: Reshape
conv_lst_m2d_5: ConvLSTM2D
reshapelstmtoconv: Reshape
flatten: Flatten
FC1: Dense
FC2: Dense
Predictions: Dense

```
Train on 4120 samples, validate on 1780 samples
Epoch 1/20
4120/4120 [==============================] - 14s 3ms/step - loss: 0.0046 -
accuracy: 0.9988 - val_loss: 0.5636 - val_accuracy: 0.8994
Epoch 2/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0655 -
accuracy: 0.9789 - val_loss: 1.1785 - val_accuracy: 0.7573
Epoch 3/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0646 -
accuracy: 0.9748 - val_loss: 0.3715 - val_accuracy: 0.8646
Epoch 4/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0318 -
accuracy: 0.9905 - val_loss: 0.7867 - val_accuracy: 0.8399
Epoch 5/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0208 -
accuracy: 0.9934 - val_loss: 0.9298 - val_accuracy: 0.8449
Epoch 6/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0120 -
accuracy: 0.9959 - val_loss: 1.8405 - val_accuracy: 0.7854
Epoch 7/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0110 -
accuracy: 0.9961 - val_loss: 1.3531 - val_accuracy: 0.8067
Epoch 8/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0234 -
accuracy: 0.9910 - val_loss: 1.9183 - val_accuracy: 0.8000
Epoch 9/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0300 -
accuracy: 0.9913 - val_loss: 0.9928 - val_accuracy: 0.6657
Epoch 10/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0319 -
accuracy: 0.9915 - val_loss: 0.6522 - val_accuracy: 0.8034
Epoch 11/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0392 -
accuracy: 0.9874 - val_loss: 1.1651 - val_accuracy: 0.7264
Epoch 12/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0122 -
accuracy: 0.9951 - val_loss: 1.4307 - val_accuracy: 0.6056
Epoch 13/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0073 -
accuracy: 0.9973 - val_loss: 1.5495 - val_accuracy: 0.5876
Epoch 14/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0207 -
accuracy: 0.9910 - val_loss: 0.7260 - val_accuracy: 0.7449
Epoch 15/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0112 -
accuracy: 0.9968 - val_loss: 1.2824 - val_accuracy: 0.8157
Epoch 16/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0024 -
accuracy: 0.9995 - val_loss: 1.0874 - val_accuracy: 0.8393
Epoch 17/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0026 -
accuracy: 0.9988 - val_loss: 1.8460 - val_accuracy: 0.8202
Epoch 18/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0034 -
accuracy: 0.9990 - val_loss: 1.4922 - val_accuracy: 0.8213
Epoch 19/20
```

```
4120/4120 [==============================] - 13s 3ms/step - loss: 8.0434e-04
- accuracy: 1.0000 - val_loss: 1.6618 - val_accuracy: 0.8118
Epoch 20/20
4120/4120 [==============================] - 13s 3ms/step - loss: 0.0215 -
accuracy: 0.9915 - val_loss: 0.2597 - val_accuracy: 0.9107
<keras.callbacks.callbacks.History at 0x7f75664d90d0>
```

The accuracy on the training set is 99.15% and the accuracy on the test set is 91.07%.

Testing the model on YouTube videos.

Again, we convert the video into frames and save them as image files.

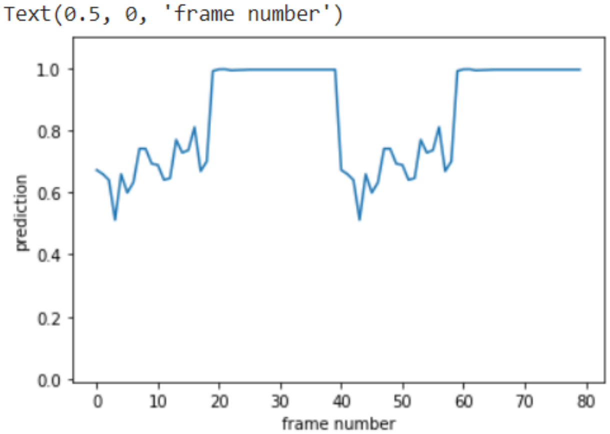
Here we call the model to predict the label of frames in the test data:

And then we plot the predicted label for each frame in the video. For this sample video, the shaking-head action is detected which was correct. For each video lips in the test set, we repeat the same process and then we upload true positive samples in the ilab website by a JSON file named CSCE636_baharealik_V3_JSONfile.

An example of a labeled Youtube video by the model is shown below:

```
plt.plot(u_test)
plt.ylim(-.01,1.1)
plt.ylabel('prediction')
plt.xlabel('frame number')
```

```
Text(0.5, 0, 'frame number')
```



Discussion:

When I was assessing detected positives of test set, I found out that the model cannot detect cutting when the knife is not visible (when the knife is held by a person right in front of the camera. But when the person holds the knife in sideways, the cutting action is recognized. Moreover, when the camera is zoomed on the cutting action, the action is not recognized anymore. In the next submission, I need to collect more data for cutting activity and reduce the false negative of the model.