Training network

December 7, 2020

[1]:
```
import os
os.getcwd()
```

[1]:
```
'C:\\Users\\Anshul\\OneDrive\\Desktop\\Gaurav'
```

[2]:
```
import cv2
import numpy as np

#extracting the frames from video
cam=cv2.VideoCapture('video_2.mp4')
input_fps=cam.get(cv2.CAP_PROP_FPS)
cam, input_fps
```

[2]:
```
(<VideoCapture 000001ED264A8450>, 29.734457193201198)
```

[3]:
```
video_length=int(cam.get(cv2.CAP_PROP_FRAME_COUNT))
video_length, input_fps
```

[3]:
```
(513, 29.734457193201198)
```

[4]:
```
ret_val, image=cam.read()
count=0
while ret_val:
    cv2.imwrite("frame_1/frame%d.png"%count, image)
    count+=1
    ret_val, image=cam.read()
```

[15]:
```
#creating the training dataset
path = './frame_1/'
new_path = './data/'
images = os.listdir(path)
images[:10]
```

[15]:
```
['frame0.png',
 'frame1.png',
 'frame10.png',
 'frame100.png',
 'frame101.png',
```

```
 'frame102.png',
 'frame103.png',
 'frame104.png',
 'frame105.png',
 'frame106.png']
```

[16]:
```
for i in range(0,54):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'g_'+'frame%d.png'%i,img)
```

[17]:
```
for i in range(54,93):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'b_'+'frame%d.png'%i,img)

for i in range(93,194):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'g_'+'frame%d.png'%i,img)

for i in range(194,230):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'b_'+'frame%d.png'%i,img)

for i in range(230,398):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'g_'+'frame%d.png'%i,img)

for i in range(398,425):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'b_'+'frame%d.png'%i,img)

for i in range(425,513):
    img = cv2.imread(path+'frame%d.png'%i)
    cv2.imwrite(new_path+'g_'+'frame%d.png'%i,img)
```

[18]:
```
images = os.listdir(new_path)

np.random.shuffle(images)
images[:10]
```

[18]:
```
['b_frame62.png',
 'g_frame269.png',
 'g_frame0.png',
 'b_frame202.png',
 'g_frame375.png',
 'g_frame232.png',
 'g_frame162.png',
 'g_frame391.png',
```

```
 'b_frame198.png',
 'g_frame292.png']
```

[20]:
```
labels = []
for i in images:
    if i[0] == 'g':
        labels.append(1)
    else:
        labels.append(0)
```

[21]:
```
labels[:10]
```

[21]:
```
[0, 1, 1, 0, 1, 1, 1, 1, 0, 1]
```

[23]:
```
#size of overall data
len(images)
```

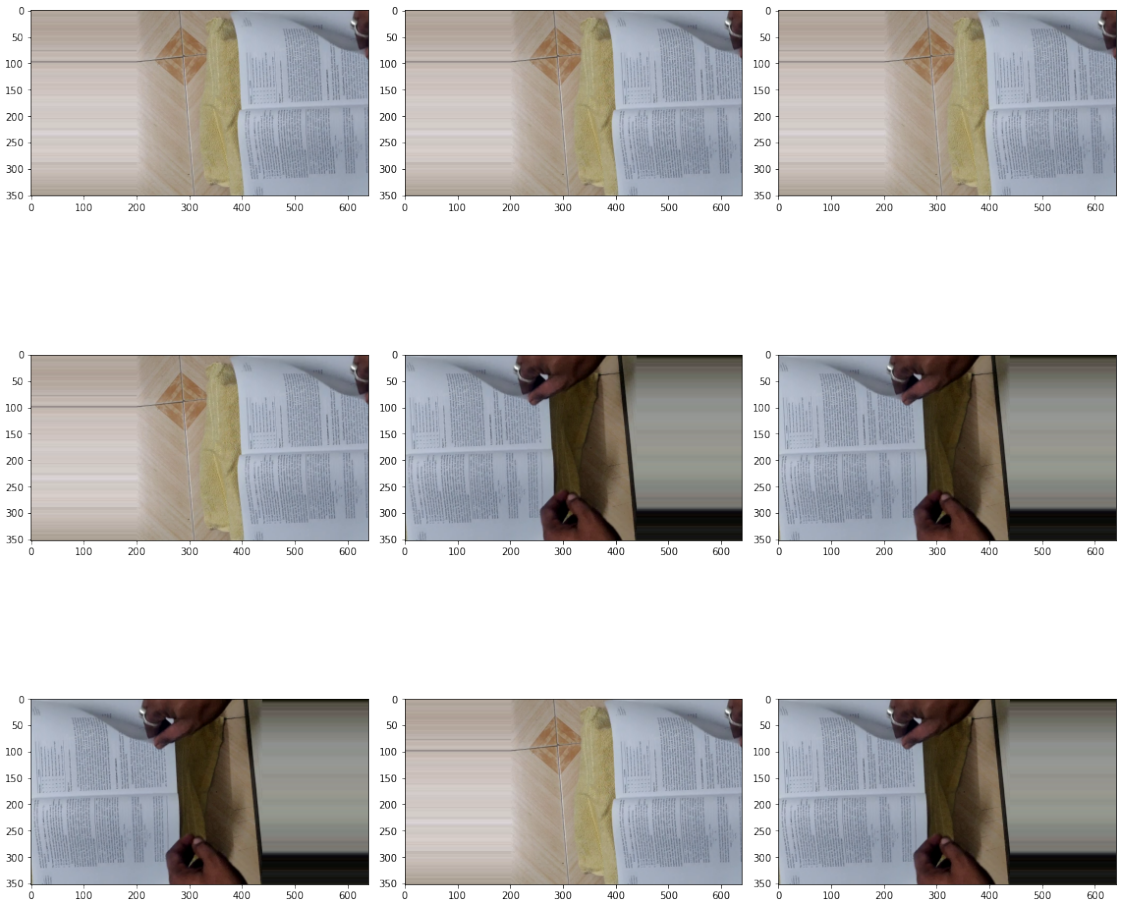
[23]:
```
513
```

[24]:
```
#Data augmentation step for better generalization
# example of horizontal shift image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
```

```
Using TensorFlow backend.
```

[32]:
```
# load the image
img = load_img(new_path+images[3])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-200,200])
# prepare iterator
it = datagen.flow(samples, batch_size=1)

fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
for i in range(3):
    for j in range(3):
        # define subplot
        # generate batch of images
        batch = it.next()
```

```
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```

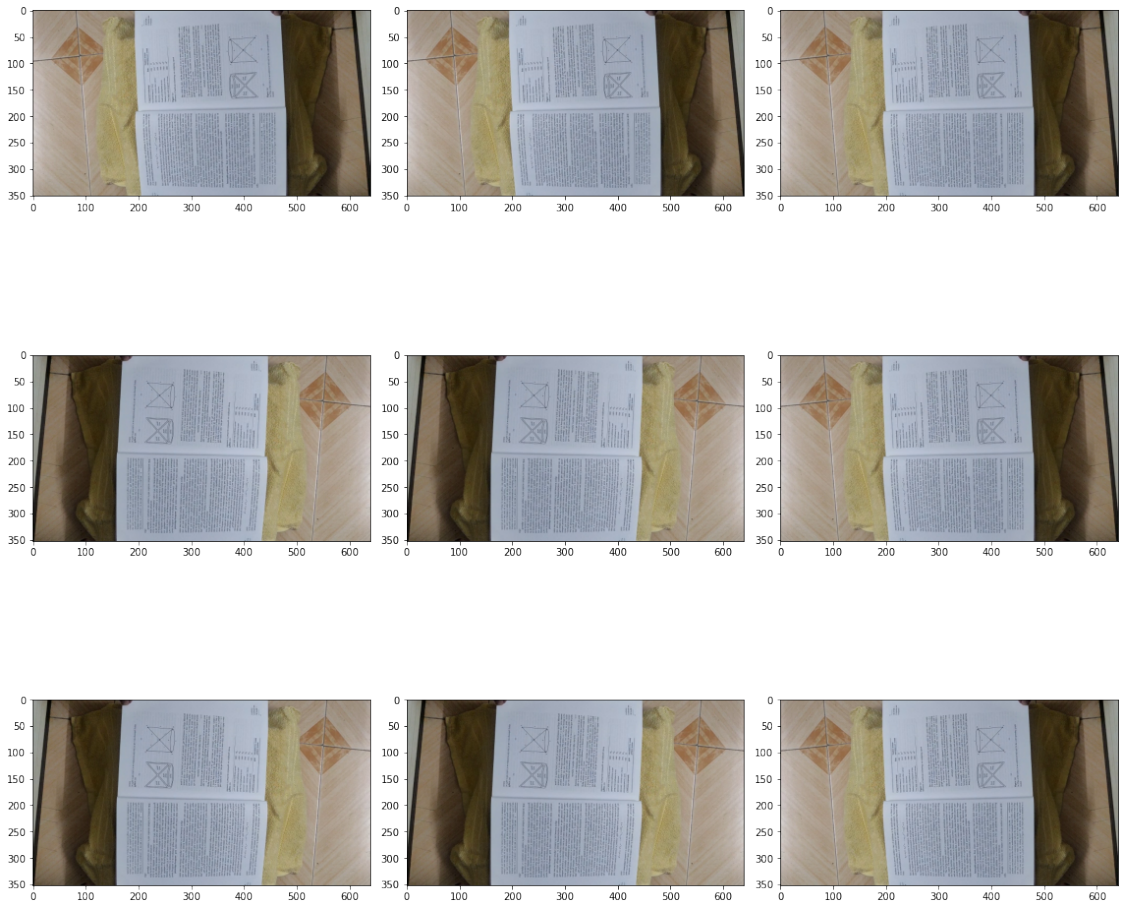


[33]:
```
img = load_img(new_path+images[9])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(height_shift_range=0.5)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
```

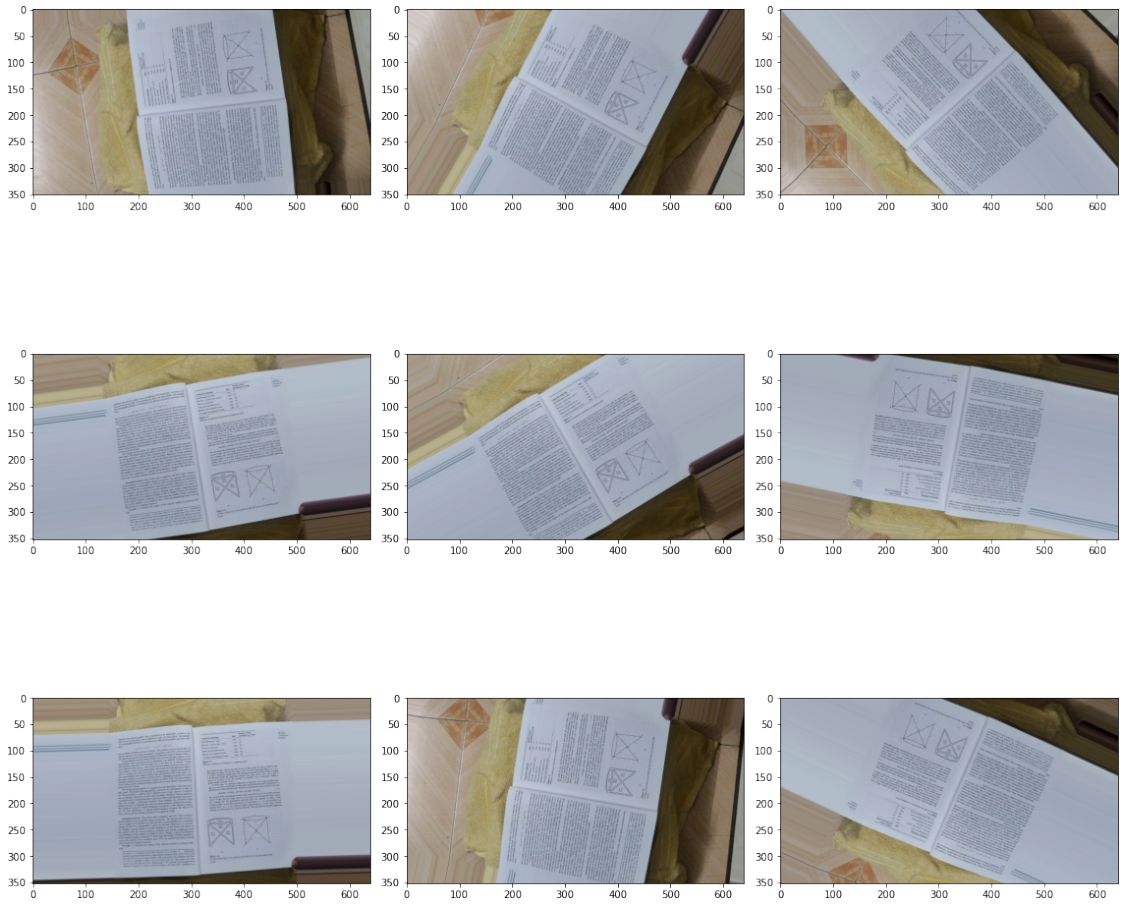
```
for i in range(3):
    for j in range(3):
        # define subplot
        # generate batch of images
        batch = it.next()
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```

[34]:
```
# load the image
img = load_img(new_path+images[9])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
```

```
# create image data augmentation generator
datagen = ImageDataGenerator(horizontal_flip=True)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
for i in range(3):
    for j in range(3):
        # define subplot
        # generate batch of images
        batch = it.next()
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```

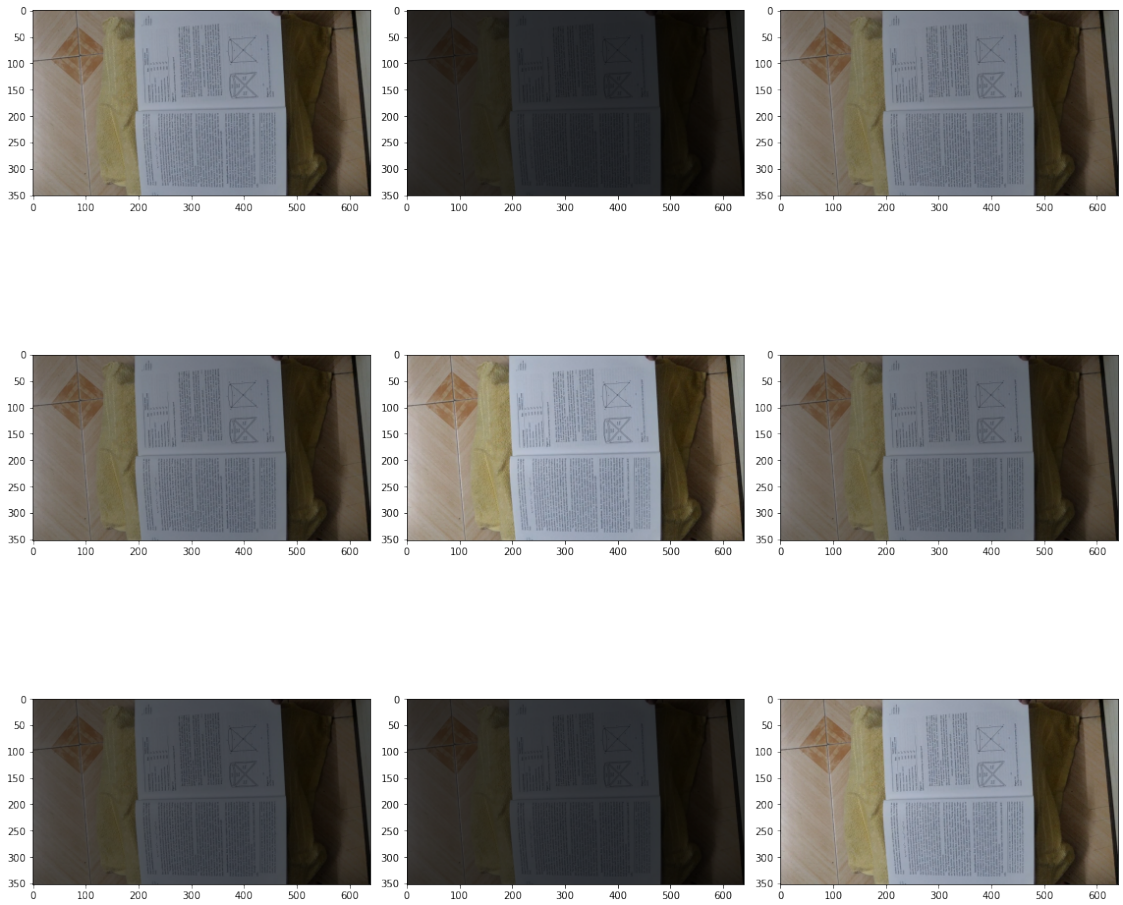


[35]:
```
# load the image
img = load_img(new_path+images[9])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(rotation_range=90)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
for i in range(3):
    for j in range(3):
        # define subplot
        # generate batch of images
        batch = it.next()
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```



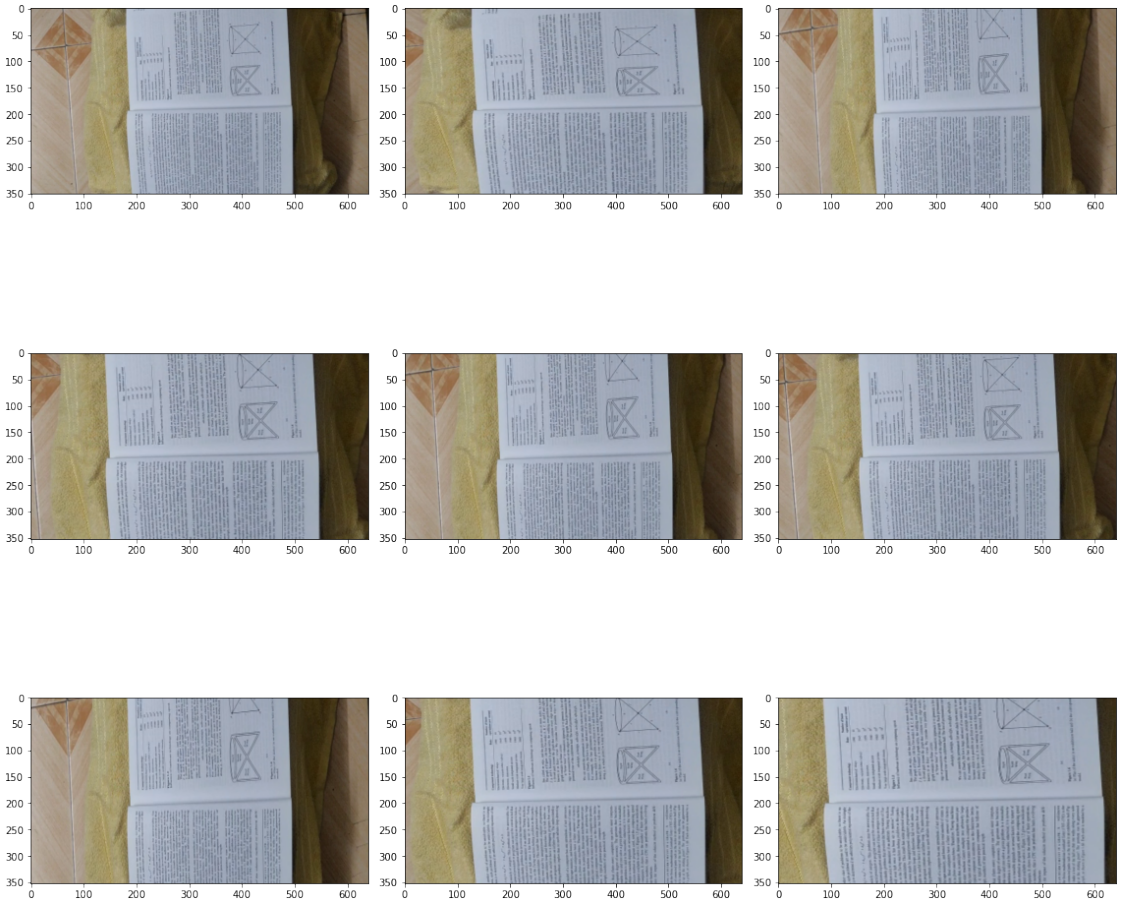
[37]:
```
# load the image
img = load_img(new_path+images[9])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(brightness_range=[0.2,1.0])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
for i in range(3):
    for j in range(3):
        # define subplot
        # generate batch of images
        batch = it.next()
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
```

```
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```



[38]:
```
img = load_img(new_path+images[9])
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(zoom_range=[0.5,1.0])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
fig, ax = pyplot.subplots(3,3,figsize=(16,16))
# generate samples and plot
for i in range(3):
    for j in range(3):
```

```
        # define subplot
        # generate batch of images
        batch = it.next()
        # convert to unsigned integers for viewing
        image = batch[0].astype('uint8')
        # plot raw pixel data
        ax[i,j].imshow(image)
# show the figure
pyplot.tight_layout()
pyplot.show()
```



[40]:
```
data = images
```

[44]:
```
images = []
for i in data:
    images.append(new_path+i)
```

[45]:
```
images[2]
```

[45]:
```
'./data/g_frame0.png'
```

[46]:
```
from sklearn.model_selection import train_test_split
#x for data, y for labels
train_x, test_x, train_y, test_y = train_test_split(images,␣
↪labels,random_state=43)
```

[48]:
```
train_x[:2], train_y[:2]
```

[48]:
```
(['./data/b_frame202.png', './data/g_frame443.png'], [0, 1])
```

[56]:
```
len(train_x)
```

[56]:
```
384
```

[49]:
```
def euclidean_distance(vects):
    '''Compute Euclidean Distance between two vectors'''
    x, y =vects
    return K.sqrt(K.sum(K.square(x-y), axis=1, keepdims=True))
```

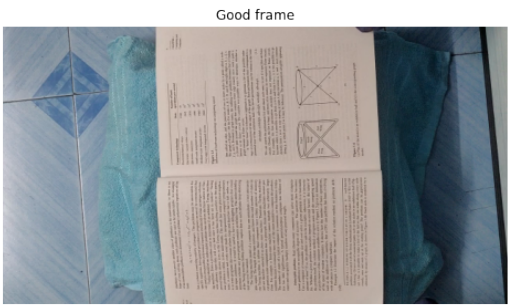
[52]:
```
def contrastive_loss(y_true, y_pred):
    margin=1
    return K.mean(y_true*K.square(y_pred)+(1-y_true)*K.square(K,␣
↪maximum(margin-y_pred,0)))
```

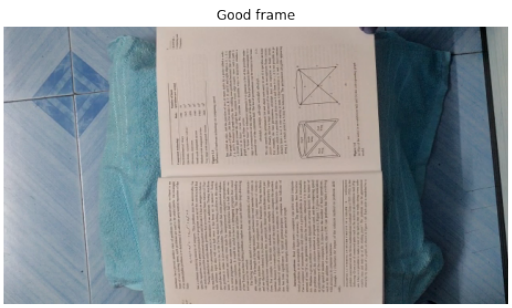
[53]:
```
#necessary modules
import sys
import numpy as np
import pickle
import os
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

import cv2
import time
import itertools
import random

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import keras
from keras import models
import tensorflow as tf
from keras.models import Sequential
from keras.optimizers import Adam, RMSprop
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate,␣
↪Dropout
```

```
from keras.models import Model

from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform

from keras.engine.topology import Layer, InputSpec
from keras.regularizers import l2
from keras import backend as K
import keras.backend.tensorflow_backend as tfback
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from keras.utils.multi_gpu_utils import multi_gpu_model
```

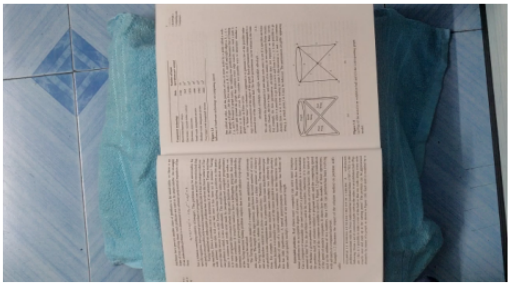
[89]:
```
def visualize_frames(j,k,images):
    im1, im2 = cv2.imread(images[j]), cv2.imread(images[k])
    fig, ax = plt.subplots(1,2,figsize=(18,18))
    ax[0].imshow(im1)
    ax[0].set_title('Good frame' if images[j][7] == 'g' else 'bad frame')
    ax[1].imshow(im2)
    ax[1].set_title('Good frame' if images[k][7] == 'g' else 'bad frame')
    ax[0].axis("off")
    ax[1].axis("off")
```

[90]:
```
for i in range(5):
    j, k = random.randint(0,len(images)),random.randint(0,len(images))
    visualize_frames(j,k,images)
```

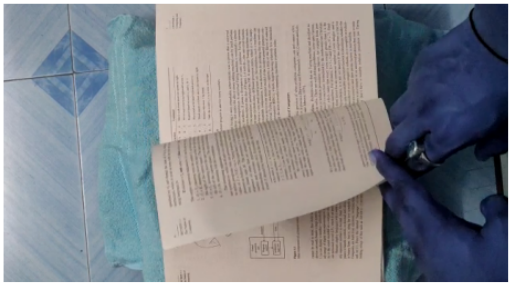




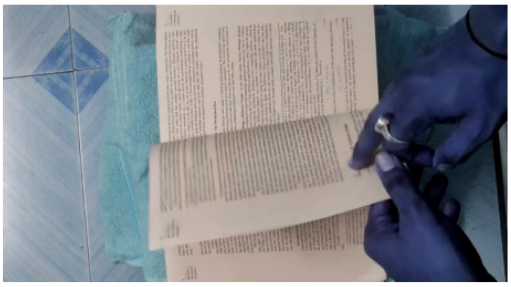
Good frame



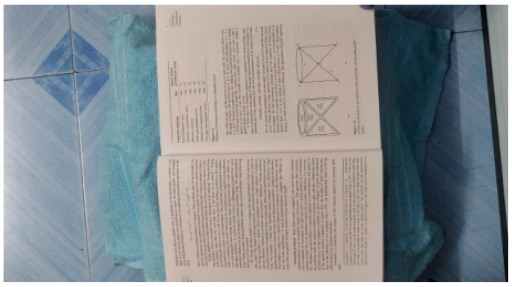
bad frame



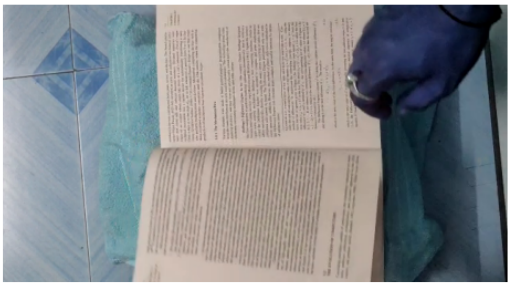
bad frame



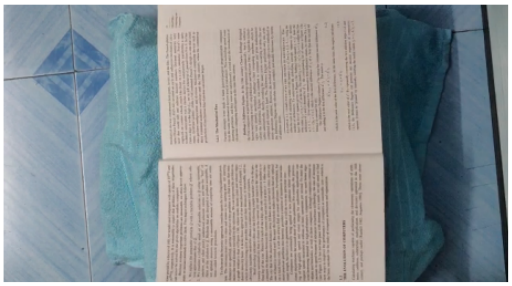
Good frame



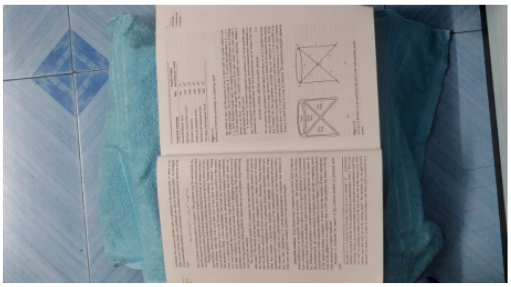
bad frame



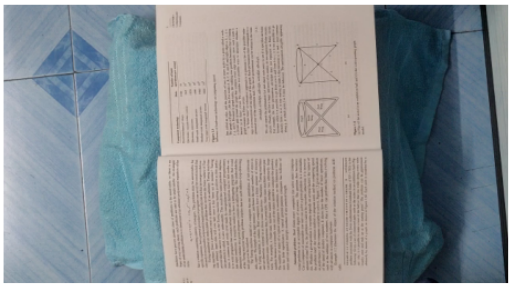
Good frame



Good frame



Good frame



[112]:
```
batch_sz = 10

num_train_samples = len(train_x)
num_val_samples =  len(val_x)
num_test_samples = len(test_x)
print(num_train_samples,num_val_samples,num_test_samples)
```

```
216 72 129
```

[115]:
```
len(train_x)
```

[115]:
```
216
```

[365]:
```
# define cnn model
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_uniform', padding='same', input_shape=(64, 128, 1)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Conv2D(64, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Conv2D(128, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu',␣
 ↪kernel_initializer='he_uniform'))
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        return model
```

[366]:
```
#input shape
input_shape=(64, 128, 1)

input_a = Input(shape=(input_shape))
#input_b = Input(shape=(input_shape))

print(input_a)
```

```
Tensor("input_20:0", shape=(None, 64, 128, 3), dtype=float32)
```

[ ]:
```

```

[367]:
```
from keras.optimizers import SGD

processed_a = base_network(input_a)

model = Model(input=[input_a], output=processed_a)

opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

```
␣
↪---------------------------------------------------------------------------

       InvalidArgumentError                      Traceback (most recent call␣
↪last)

       ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\ops.py in␣
↪_create_c_op(graph, node_def, inputs, control_inputs)
      1618   try:
   -> 1619     c_op = c_api.TF_FinishOperation(op_desc)
      1620   except errors.InvalidArgumentError as e:


       InvalidArgumentError: Depth of output (40) is not a multiple of the␣
↪number of groups (3) for 'sequential_12_3/conv2d_31/convolution' (op:␣
↪'Conv2D') with input shapes: [?,64,128,3], [7,7,1,40].


   During handling of the above exception, another exception occurred:


       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-367-6caac35683d5> in <module>
         1 from keras.optimizers import SGD
         2
   ----> 3 processed_a = base_network(input_a)
         4
         5 model = Model(input=[input_a], output=processed_a)


       ~\.conda\envs\env1\lib\site-packages\keras\backend\tensorflow_backend.py␣
↪in symbolic_fn_wrapper(*args, **kwargs)
        73         if _SYMBOLIC_SCOPE.value:
        74             with get_graph().as_default():
```

```
---> 75                 return func(*args, **kwargs)
     76         else:
     77             return func(*args, **kwargs)


~\.conda\envs\env1\lib\site-packages\keras\engine\base_layer.py in␣
↪__call__(self, inputs, **kwargs)
    487             # Actually call the layer,
    488             # collecting output(s), mask(s), and shape(s).
--> 489             output = self.call(inputs, **kwargs)
    490             output_mask = self.compute_mask(inputs, previous_mask)
    491


~\.conda\envs\env1\lib\site-packages\keras\engine\network.py in␣
↪call(self, inputs, mask)
    581             return self._output_tensor_cache[cache_key]
    582         else:
--> 583             output_tensors, _, _ = self.run_internal_graph(inputs,␣
↪masks)
    584             return output_tensors
    585


~\.conda\envs\env1\lib\site-packages\keras\engine\network.py in␣
↪run_internal_graph(self, inputs, masks)
    738                                     kwargs['mask'] = computed_mask
    739                             output_tensors = to_list(
--> 740                                 layer.call(computed_tensor,␣
↪**kwargs))
    741                             output_masks = layer.
↪compute_mask(computed_tensor,
    742                                                                 ␣
↪computed_mask)


~\.conda\envs\env1\lib\site-packages\keras\layers\convolutional.py in␣
↪call(self, inputs)
    169                 padding=self.padding,
    170                 data_format=self.data_format,
--> 171                 dilation_rate=self.dilation_rate)
    172         if self.rank == 3:
    173             outputs = K.conv3d(


~\.conda\envs\env1\lib\site-packages\keras\backend\tensorflow_backend.py␣
↪in conv2d(x, kernel, strides, padding, data_format, dilation_rate)
```

```
      3715         padding=padding,
      3716         data_format=tf_data_format,
   -> 3717         **kwargs)
      3718     if data_format == 'channels_first' and tf_data_format == 'NHWC':
      3719         x = tf.transpose(x, (0, 3, 1, 2))  # NHWC -> NCHW


       ~\.conda\envs\env1\lib\site-packages\tensorflow_core\python\ops\nn_ops.
↪py in convolution_v2(input, filters, strides, padding, data_format, dilations,␣
↪name)
       916       data_format=data_format,
       917       dilations=dilations,
   --> 918       name=name)
       919
       920


       ~\.conda\envs\env1\lib\site-packages\tensorflow_core\python\ops\nn_ops.
↪py in convolution_internal(input, filters, strides, padding, data_format,␣
↪dilations, name, call_from_convolution)
      1008           data_format=data_format,
      1009           dilations=dilations,
   -> 1010           name=name)
      1011     else:
      1012       if channel_index == 1:


       ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\ops\gen_nn_ops.py in␣
↪conv2d(input, filter, strides, padding, use_cudnn_on_gpu, explicit_paddings,␣
↪data_format, dilations, name)
       966                   padding=padding, use_cudnn_on_gpu=use_cudnn_on_gpu,
       967                   explicit_paddings=explicit_paddings,
   --> 968                   data_format=data_format, dilations=dilations,␣
↪name=name)
       969   _result = _outputs[:]
       970   if _execute.must_record_gradient():


       ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\op_def_library.
↪py in _apply_op_helper(op_type_name, name, **keywords)
       740       op = g._create_op_internal(op_type_name, inputs, dtypes=None,
       741                                  name=scope, input_types=input_types,
   --> 742                                  attrs=attr_protos, op_def=op_def)
       743
```

```
      744     # `outputs` is returned as a separate return value so that the␣
↪output


      ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\func_graph.
↪py in _create_op_internal(self, op_type, inputs, dtypes, input_types, name,␣
↪attrs, op_def, compute_device)
      593     return super(FuncGraph, self)._create_op_internal(  # pylint:␣
↪disable=protected-access
      594         op_type, inputs, dtypes, input_types, name, attrs, op_def,
  --> 595         compute_device)
      596
      597   def capture(self, tensor, name=None, shape=None):


      ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\ops.py in␣
↪_create_op_internal(self, op_type, inputs, dtypes, input_types, name, attrs,␣
↪op_def, compute_device)
     3320           input_types=input_types,
     3321           original_op=self._default_original_op,
  -> 3322           op_def=op_def)
     3323       self._create_op_helper(ret, compute_device=compute_device)
     3324     return ret


      ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\ops.py in␣
↪__init__(self, node_def, g, inputs, output_types, control_inputs, input_types,␣
↪original_op, op_def)
     1784           op_def, inputs, node_def.attr)
     1785       self._c_op = _create_c_op(self._graph, node_def,␣
↪grouped_inputs,
  -> 1786                                 control_input_ops)
     1787       name = compat.as_str(node_def.name)
     1788     # pylint: enable=protected-access


      ~\.
↪conda\envs\env1\lib\site-packages\tensorflow_core\python\framework\ops.py in␣
↪_create_c_op(graph, node_def, inputs, control_inputs)
     1620   except errors.InvalidArgumentError as e:
     1621     # Convert to ValueError for backwards compatibility.
  -> 1622     raise ValueError(str(e))
     1623
     1624   return c_op
```

```
        ValueError: Depth of output (40) is not a multiple of the number of␣
 ↪groups (3) for 'sequential_12_3/conv2d_31/convolution' (op: 'Conv2D') with␣
 ↪input shapes: [?,64,128,3], [7,7,1,40].
```

[207]:
```
model.summary()
```

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        (None, 64, 128, 1)        0
_________________________________________________________________
sequential_12 (Sequential)   (None, 2)                 78052
=================================================================
Total params: 78,052
Trainable params: 78,052
Non-trainable params: 0
_________________________________________________________________
```

[208]:
```
# Using Keras Callbacks, save the model after every epoch
# Reduce the learning rate by a factor of 0.1 if the validation loss does not␣
 ↪improve for 5 epochs
# Stop the training using early stopping if the validation loss does not␣
 ↪improve for 12 epochs
callbacks = [
    EarlyStopping(patience=12, verbose=1),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.
 ↪000001, verbose=1),
    ModelCheckpoint('./Weights{epoch:03d}.h5', verbose=1,␣
 ↪save_weights_only=True)
]
```

[209]:
```
batch_sz = 10

num_train_samples = len(train_x)

num_val_samples =  len(val_x)
num_test_samples = len(test_x)
num_train_samples, num_val_samples, num_test_samples
```

[209]:
```
(216, 72, 129)
```

[223]:
```
a = np.zeros((3,2),dtype=np.int)
a[2][0] = 1
```

```
a
```

[223]:
```
array([[0, 0],
       [0, 0],
       [1, 0]])
```

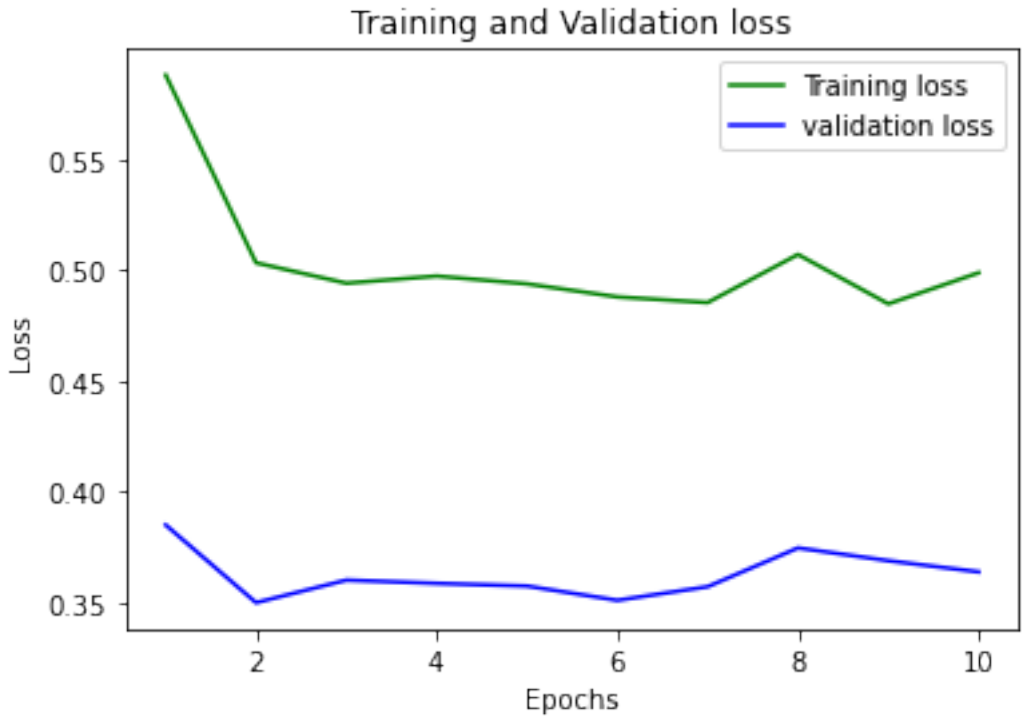
[226]:
```
def generate_batch(data_groups, lables,batch_size=10):
    '''Function to generate a batch of data with batch_size number of data␣
 ↪points'''
    img_h, img_w = 64, 128

    while True:
        k = 0
        batch=np.zeros((batch_size, img_h, img_w, 1))
        targets=np.zeros((batch_size,2),dtype=np.int)
        for ix, image in enumerate(data_groups):
            img1 = cv2.imread(image, 0)
            img1 = cv2.resize(img1, (img_w, img_h))
            img1 = np.array(img1, dtype = np.float64)
            img1 /= 255
            img1 = img1[..., np.newaxis]
            batch[k, :, :, :] = img1
            temp = lables[ix]
            if temp:
                targets[k][1]=1
            else:
                targets[k][0]=1
            k += 1
            if k == batch_size:
                yield batch, targets
                k = 0
                batch=np.zeros((batch_size, img_h, img_w, 1))
                targets=np.zeros((batch_size,2),dtype=np.int)
```

[227]:
```
results = model.fit_generator(generate_batch(train_x,train_y),
                              steps_per_epoch = num_train_samples//batch_sz,
                              epochs = 10,
                              validation_data = generate_batch(val_x,val_y),
                              validation_steps = num_val_samples//batch_sz,
                              callbacks = callbacks)
```

```
Epoch 1/10
21/21 [==============================] - 26s 1s/step - loss: 0.5880 - accuracy:
0.8048 - val_loss: 0.3849 - val_accuracy: 0.8000
```

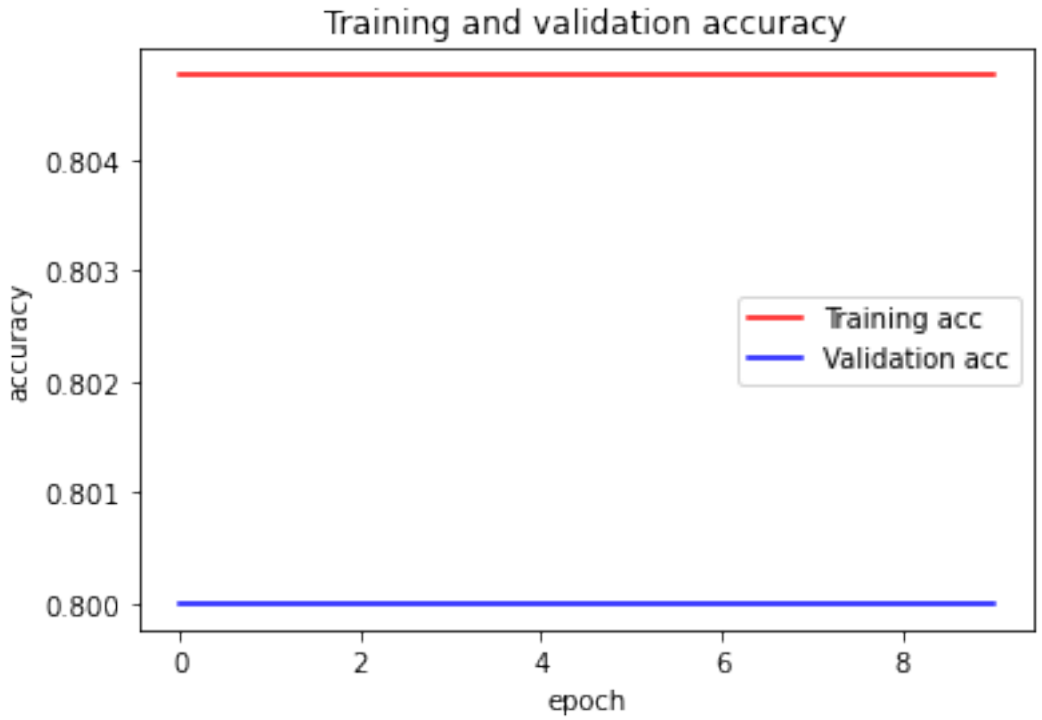
```
Epoch 00001: saving model to ./Weights001.h5
Epoch 2/10
21/21 [==============================] - 11s 547ms/step - loss: 0.5032 -
accuracy: 0.8048 - val_loss: 0.3498 - val_accuracy: 0.8000

Epoch 00002: saving model to ./Weights002.h5
Epoch 3/10
21/21 [==============================] - 23s 1s/step - loss: 0.4940 - accuracy:
0.8048 - val_loss: 0.3600 - val_accuracy: 0.8000

Epoch 00003: saving model to ./Weights003.h5
Epoch 4/10
21/21 [==============================] - 23s 1s/step - loss: 0.4972 - accuracy:
0.8048 - val_loss: 0.3585 - val_accuracy: 0.8000

Epoch 00004: saving model to ./Weights004.h5
Epoch 5/10
21/21 [==============================] - 14s 661ms/step - loss: 0.4937 -
accuracy: 0.8048 - val_loss: 0.3573 - val_accuracy: 0.8000

Epoch 00005: saving model to ./Weights005.h5
Epoch 6/10
21/21 [==============================] - 9s 425ms/step - loss: 0.4878 -
accuracy: 0.8048 - val_loss: 0.3509 - val_accuracy: 0.8000

Epoch 00006: saving model to ./Weights006.h5
Epoch 7/10
21/21 [==============================] - 8s 404ms/step - loss: 0.4853 -
accuracy: 0.8048 - val_loss: 0.3570 - val_accuracy: 0.8000

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00007: saving model to ./Weights007.h5
Epoch 8/10
21/21 [==============================] - 8s 402ms/step - loss: 0.5070 -
accuracy: 0.8048 - val_loss: 0.3745 - val_accuracy: 0.8000

Epoch 00008: saving model to ./Weights008.h5
Epoch 9/10
21/21 [==============================] - 8s 402ms/step - loss: 0.4847 -
accuracy: 0.8048 - val_loss: 0.3687 - val_accuracy: 0.8000

Epoch 00009: saving model to ./Weights009.h5
Epoch 10/10
21/21 [==============================] - 9s 425ms/step - loss: 0.4987 -
accuracy: 0.8048 - val_loss: 0.3637 - val_accuracy: 0.8000

Epoch 00010: saving model to ./Weights010.h5
```

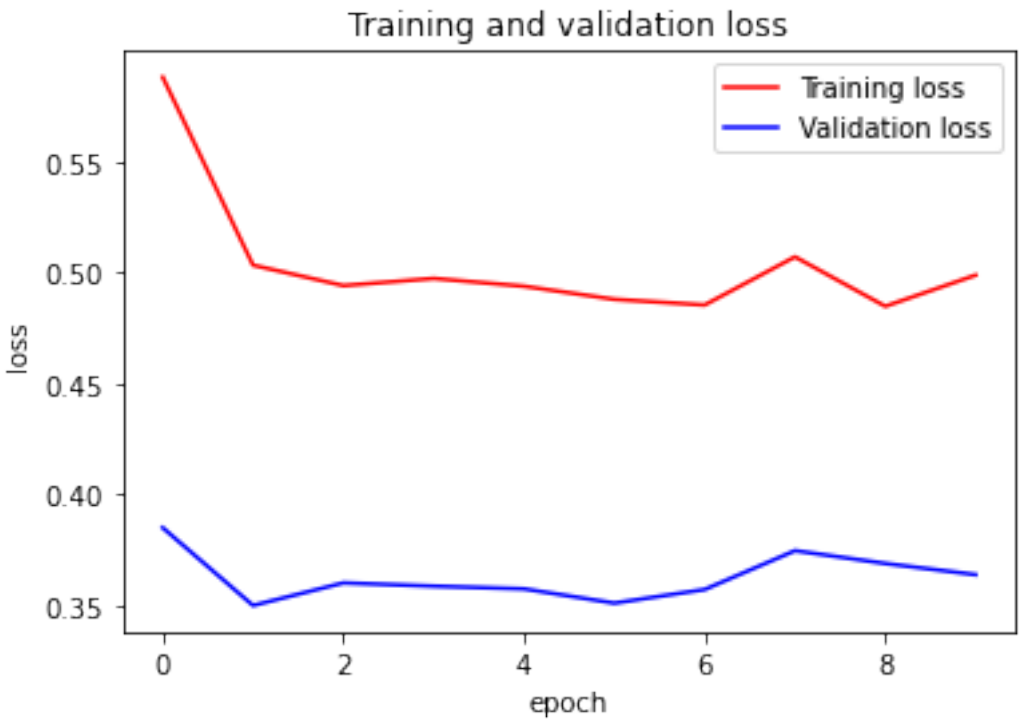
[231]:
```
loss_train = results.history['loss']
loss_val = results.history['val_loss']
epochs = range(1,14)
plt.plot(range(1,11), loss_train, 'g', label='Training loss')
plt.plot(range(1,11), loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[236]:
```
#Graphing our training and validation
acc = results.history['accuracy']
val_acc = results.history['val_accuracy']
loss = results.history['loss']
val_loss = results.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
```

```
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```





[243]:
```
# Load the weights from the epoch which gave the best validation accuracy


def load_and_check_model(weight):
    model.load_weights(weight)

    val_gen = generate_batch(val_x,val_y,1)
    pred, tr_y = [], []
    for i in range(5):
        (img1), label = next(val_gen)
        tr_y.append(label)
        pred.append(model.predict(img1)[0][0])

    tr_acc, threshold = compute_accuracy_roc(np.array(pred), np.array(tr_y))
    return tr_acc, threshold
```

[247]:
```
acc_thresh = []
for i in range(1,7,1):
    acc_thresh.append(load_and_check_model('./Weights'+str(i)+'.h5'))
    print('For model '+str(i))
    print(acc_thresh[-1])
```

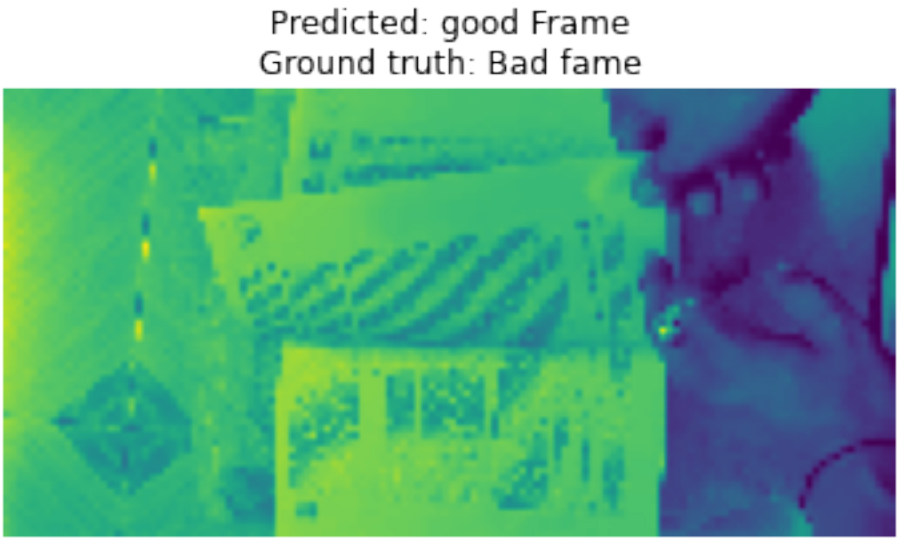
```
For model 1
```

```
(0.5, 0.23468343913555145)
For model 2
(0.5, 0.18220952153205872)
For model 3
(0.5, 0.19899199903011322)
For model 4
(0.5, 0.19652581214904785)
For model 5
(0.5, 0.19495736062526703)
For model 6
(0.5, 0.18489700555801392)
```

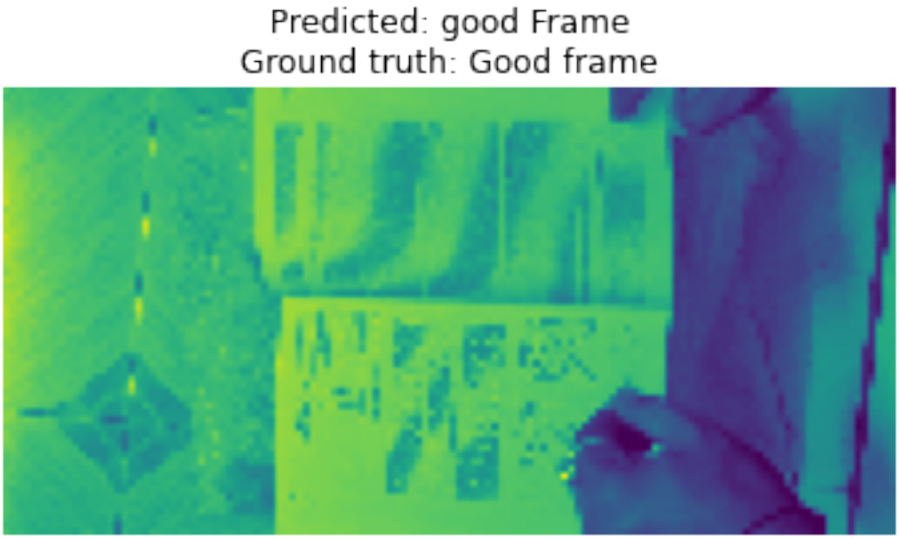
[351]:
```
def predict(data, weight='Weights/Weights10.h5'):
    model.load_weights(weight)

    images, labels = data

    result = model.predict(images)
    fig, ax = plt.subplots(1,1)
    print(labels[0][0])
    truth = 'Good frame' if labels[0][1] else 'Bad fame'
    ground = "good Frame" if result[0][1] > result[0][0] else 'Bad frame'
    print(result,result.shape)
    ax.imshow(images[0][::-1])
    ax.axis("off")
    ax.set_title('Predicted: '+ground+"\nGround truth: "+truth)
    plt.show()
```

[352]:
```
data = generate_batch(test_x,test_y,1)
for i in range(5):
    predict(next(data))
```

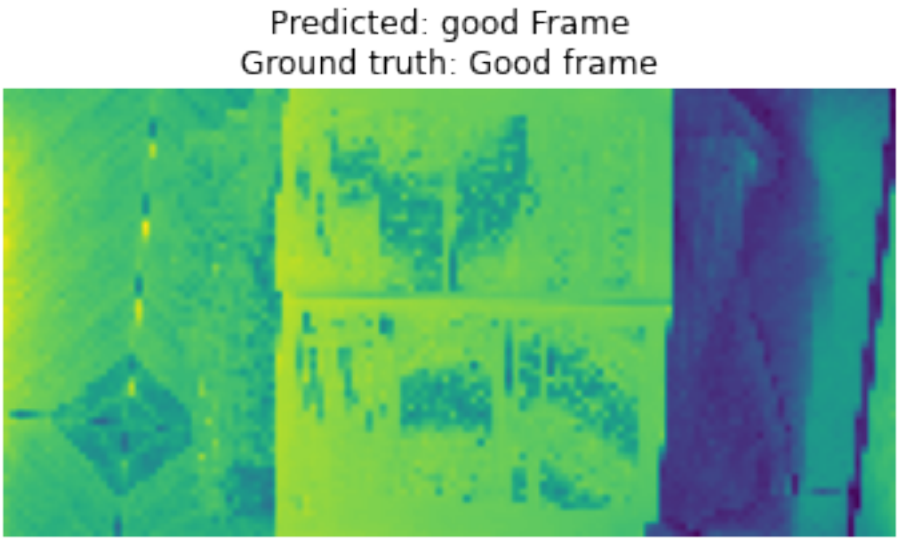
```
1
[[0.21467993 0.78532016]] (1, 2)
```



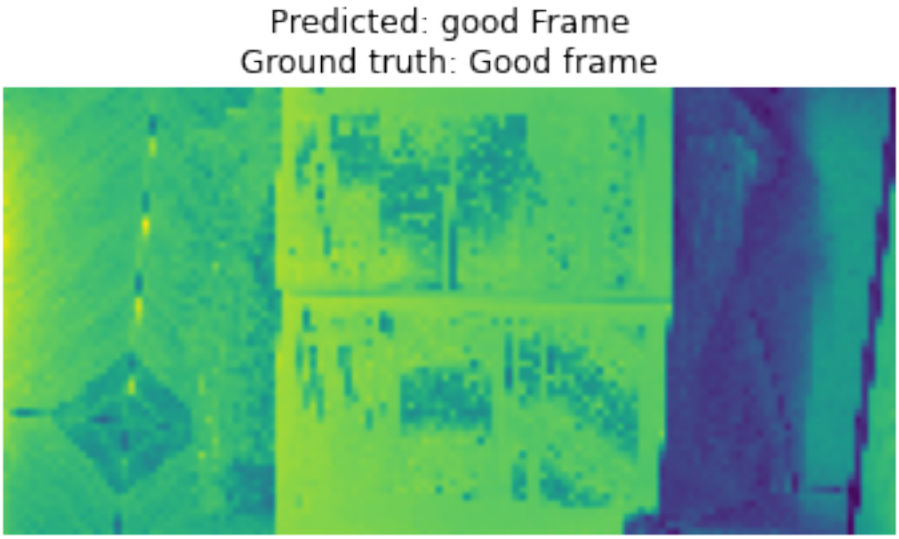

```
0
[[0.20963141 0.7903687 ]] (1, 2)
```



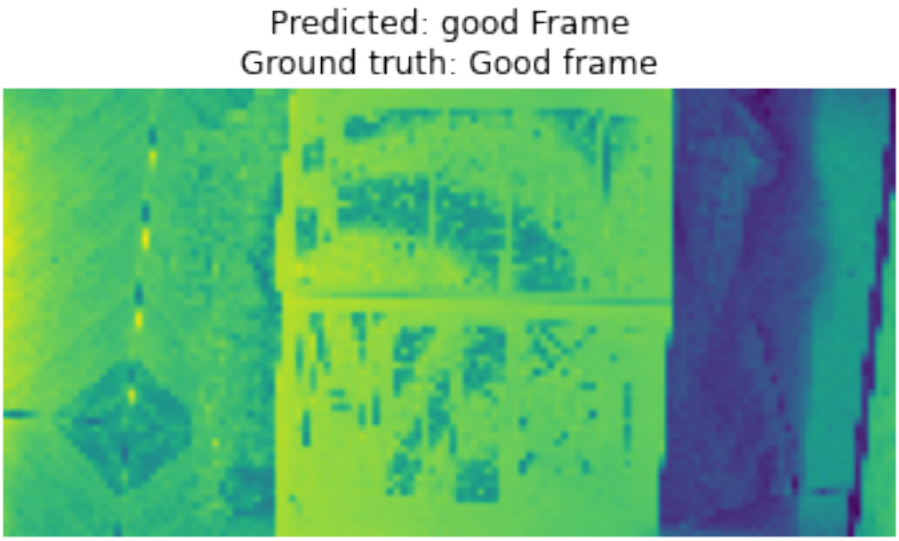

```
0
[[0.20720004 0.79279995]] (1, 2)
```




```
0
[[0.2067842 0.7932158]] (1, 2)
```



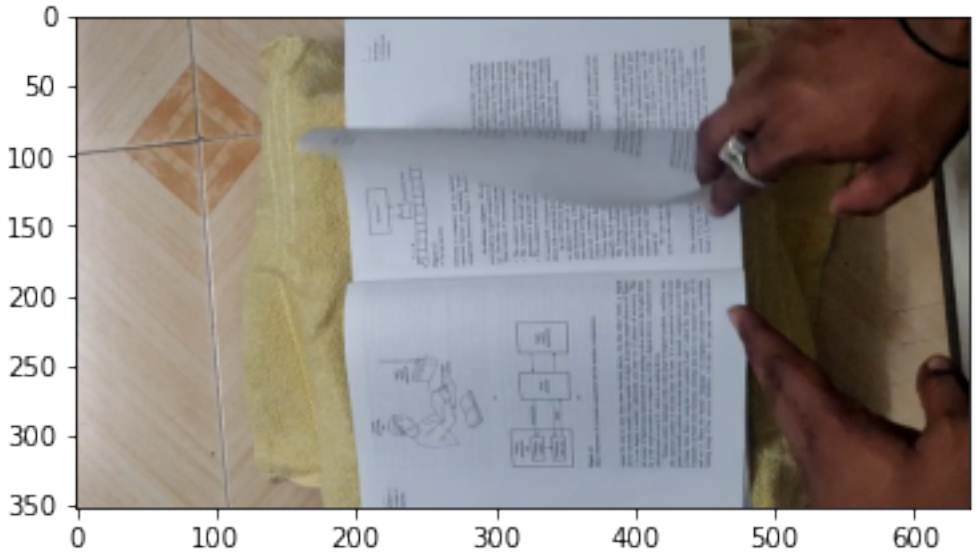

```
0
[[0.20502613 0.7949739 ]] (1, 2)
```



problem with CNN is that it need more generalize and large data in order to get better accuracy, so trying ip techniques

[364]:
```
x = cv2.imread(images[0])
x = x[:,:,::-1]
f, a = plt.subplots()
a.imshow(x)
```

[364]:
```
<matplotlib.image.AxesImage at 0x1ed4d5e48c8>
```



[368]:
```
#extracting the frames from test video
cam=cv2.VideoCapture('test_1.mp4')
input_fps=cam.get(cv2.CAP_PROP_FPS)
cam, input_fps
```

[368]:
```
(<VideoCapture 000001ED4C0BA3F0>, 29.98624834470816)
```

[369]:
```
video_length=int(cam.get(cv2.CAP_PROP_FRAME_COUNT))
video_length, input_fps
```

[369]:
```
(628, 29.98624834470816)
```

[370]:
```
ret_val, image=cam.read()
count=0
while ret_val:
    cv2.imwrite("test/frame%d.png"%count, image)
    count+=1
    ret_val, image=cam.read()
```

[386]:
```
def good_or_not(image, weight='Weights/Weights10.h5'):
    img_h, img_w = 64, 128
    model.load_weights(weight)
    img1 = cv2.imread('test/'+image, 0)
    img1 = cv2.resize(img1, (img_w, img_h))
    img1 = np.array(img1, dtype = np.float64)
    img1 /= 255
    img1 = img1[..., np.newaxis]
    img1 = img1[np.newaxis]

    result = model.predict(img1)

    return 1 if result[0][1] > result[0][0] else 0
```

[390]:
```
images_check = os.listdir('test/')
good_frames = []

print("Total Images for testing new: ",len(images_check))
```

```
Total Images for testing new:  14
```

[396]:
```
good_frames = []
for i in images_check:
    if good_or_not(i):
        good_frames.append(i)
```

```
print("Images After removal of bed frames: ",len(good_frames))
```

```
Images After removal of bed frames:  11
```

[461]:
```
def binarization(images):
    for i in images:
        print(i)
        img = cv2.imread('test/'+i)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        print(img.shape)

        ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
        gray = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.
↪THRESH_OTSU)[1]
        #ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
        #ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
        #ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
        #ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)

        """
        f, a = plt.subplots(2,2,dpi=400,figsize=(20,20))
        c = 0
        th = [thresh1[:,:,::-1],thresh2[:,:,::-1],thresh3[:,:,::-1],thresh4[:,:
↪,::-1]]
        for i in range(2):
            for j in range(2):
                a[i,j].imshow(th[c])
                c+= 1
                a[i,j].axis("off")

        plt.show()
        """
        # the window showing output images
        # with the corresponding thresholding
        # techniques applied to the input images
        cv2.imshow('Binary Threshold', thresh1)
        #cv2.imshow("binary",gray)
        #cv2.imshow('Binary Threshold Inverted', thresh2)
        #cv2.imshow('Truncated Threshold', thresh3)
        #cv2.imshow('Set to 0', thresh4)
        #cv2.imshow('Set to 0 Inverted', thresh5)
        # De-allocate any associated memory usage
        if cv2.waitKey(0) & 0xff == 27:
            cv2.destroyAllWindows()
```

[474]:
```
binarization([good_frames[2]])
```

```
frame146.png
(1080, 1920)
```

[464]:
```
!pip install pytesseract
```

```
Collecting pytesseract
  Downloading pytesseract-0.3.6.tar.gz (13 kB)
Requirement already satisfied: Pillow in
c:\users\anshul\.conda\envs\env1\lib\site-packages (from pytesseract) (8.0.1)
Building wheels for collected packages: pytesseract
  Building wheel for pytesseract (setup.py): started
  Building wheel for pytesseract (setup.py): finished with status 'done'
  Created wheel for pytesseract: filename=pytesseract-0.3.6-py2.py3-none-any.whl
size=13635
sha256=6a527cbda65f583b5138f9d02af5f35f3f5cbfa5d80bf7908ff2cce0ef021f96
  Stored in directory: c:\users\anshul\appdata\local\pip\cache\wheels\f1\2f\a5\5
74c57fb22cfcf24f315c8feda132fd0463a9b07ef78394d07
Successfully built pytesseract
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.6
```

[465]:
```
from PIL import Image
import pytesseract
```

[473]:
```
pytesseract
```

[473]:
```
<module 'pytesseract' from 'C:\\Users\\Anshul\\.conda\\envs\\env1\\lib\\site-
packages\\pytesseract\\__init__.py'>
```

[483]:
```
pytesseract.pytesseract.tesseract_cmd =  r'C:\Program␣
↪Files\Tesseract-OCR\tesseract.exe'
```

[484]:
```
text = pytesseract.image_to_string(Image.open(os.
↪getcwd()+'\\test\\'+good_frames[2]))
```

[489]:
```
text.split("\n")
```

[489]:
```
['nal Requirement',
 '17.1 User Class 1',
 '1.72 User Class 2',
 '1.7.3 User Class3',
 'CHAPTER 2 DESIGN: Analysis, Design Methodology and Implementat',
 '2.1 Canvas Description',
 '2.1.1 AEIOU Canvas',
 '21.2 Ideation Canvas',
```

```
 '2.1.3 Product Development Canvas',
 '2.1.4 Empathy Mapping Canvas',
 'ML Diagram',
 '1 Use Case Dingram',
 '22 Class Diagram',
 '3 Sequence Diagram',
 '24 Activity Diagram',
 'CHAPTER 3 IMPLEMENTATION MODEL.',
 '| Registration Process',
 'in Page',
 '',
 '3 3 Forget Password Process',
 '',
 '3.4 Ul Design',
 '',
 '3.5 Assumptions and Dependencies',
 '',
 ' ',
 '\x0c']
```

[490]:
```
len(text)
```

[490]: 489

[494]:
```
#manually calculating text word accuracy is
len(text) / 560 *100
```

[494]: 87.32142857142857

[497]:
```
#word level accuracy
len(text.split(" ")) / 68 *  100
```

[497]: 80.88235294117648

[ ]:
```

```