Figure Skating Video Classification

by Elena Golubeva and Vladislav Goteiner

01 ABOUT THE PROJECT

02 PROCESS

03 MODEL STRUCTURE

04 CODE EXAMPLE

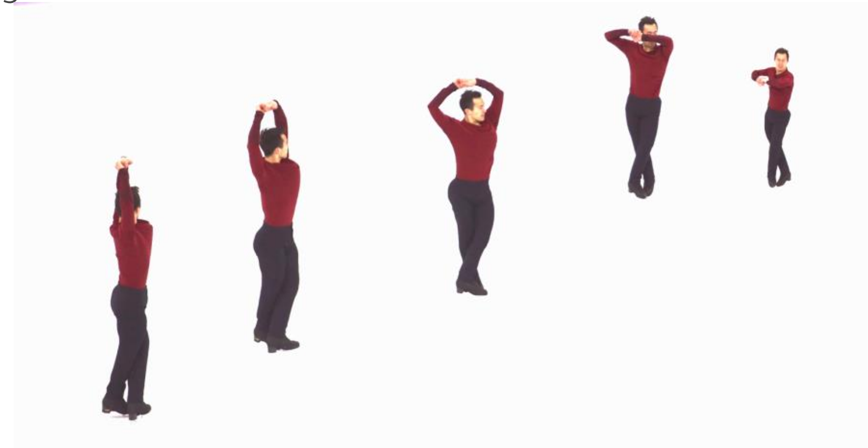
05 FINAL RESULTS

01

FIGURE SKATING JUMP CLASSIFICATION

About the project:

Tell what kind of jump is on the given video. What score would figure skater get for his element execution.

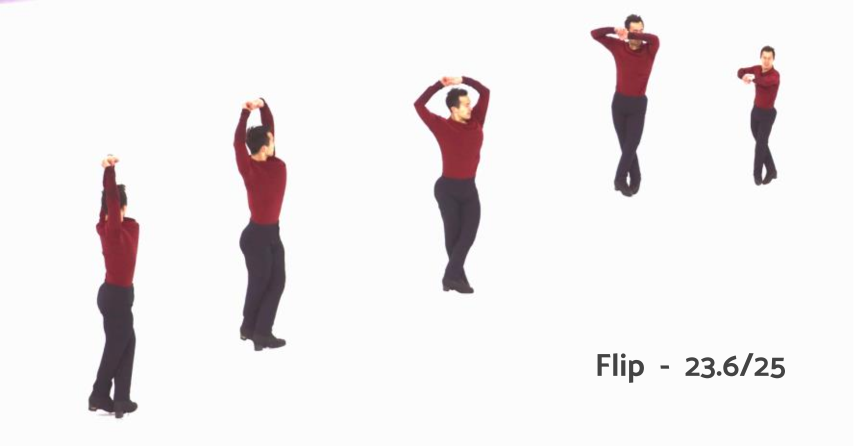


01

FIGURE SKATING JUMP CLASSIFICATION

Expectations:

To classify element with over 85% accuracy. Give a score based on similar element executions that we learned.



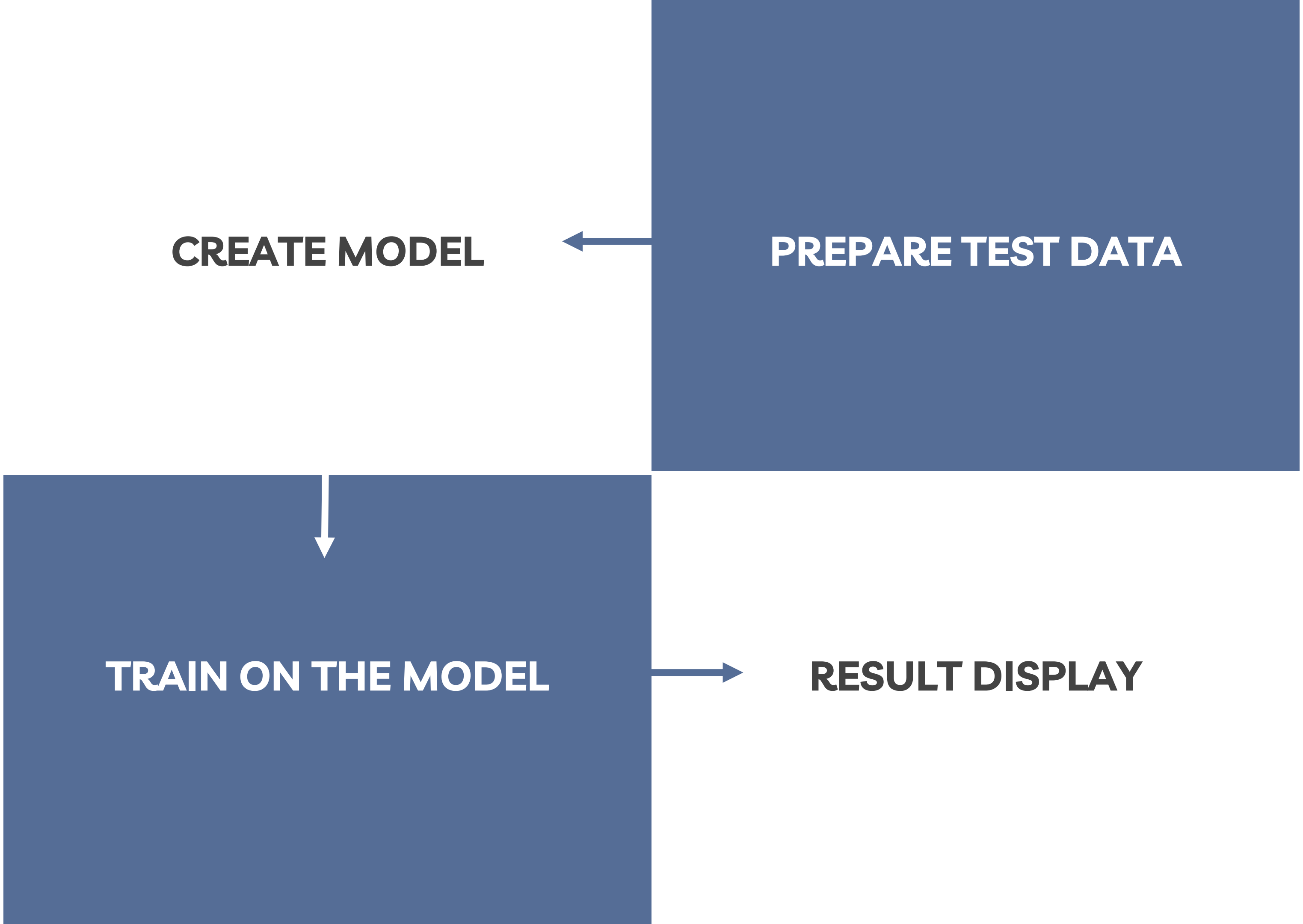
01

FIGURE SKATING JUMP CLASSIFICATION

Obstacles:

Examples from different classes has very little differences.

02



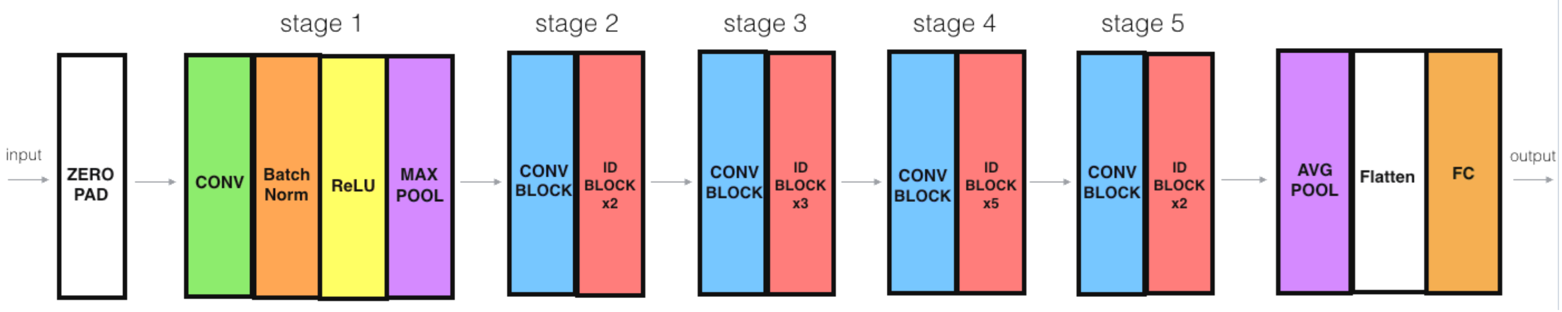
PREPARE TEST DATA

CREATE MODEL

TRAIN ON THE MODEL

RESULT DISPLAY

03



ResNet50

Average Pooling 2D

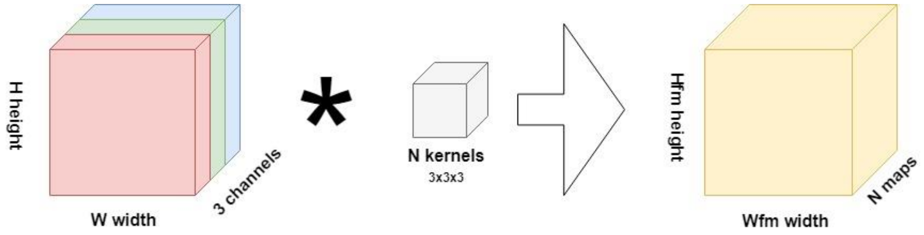
Flatten

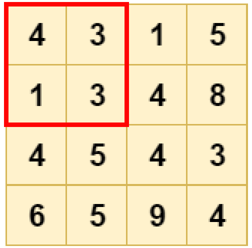
Relu x2

Dropout

Softmax

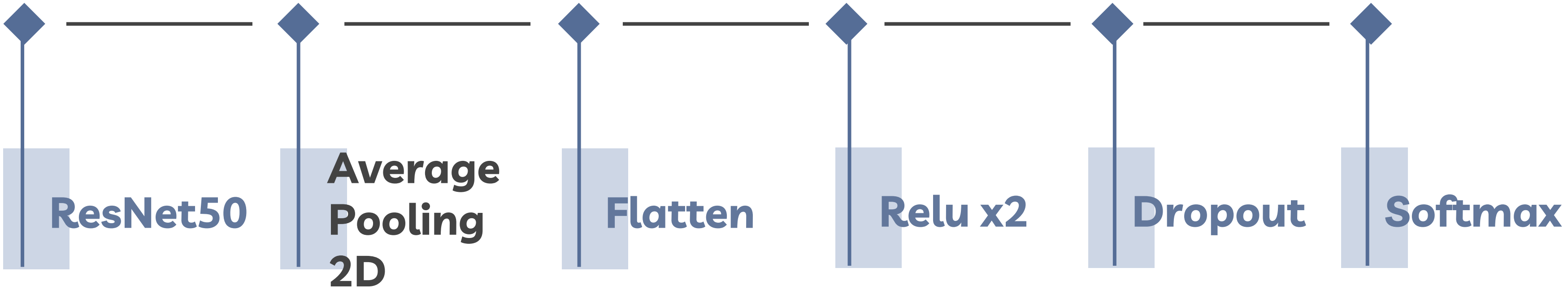
03





4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

Avg([4, 3, 1, 3]) = 2.75



ResNet50

Average Pooling 2D

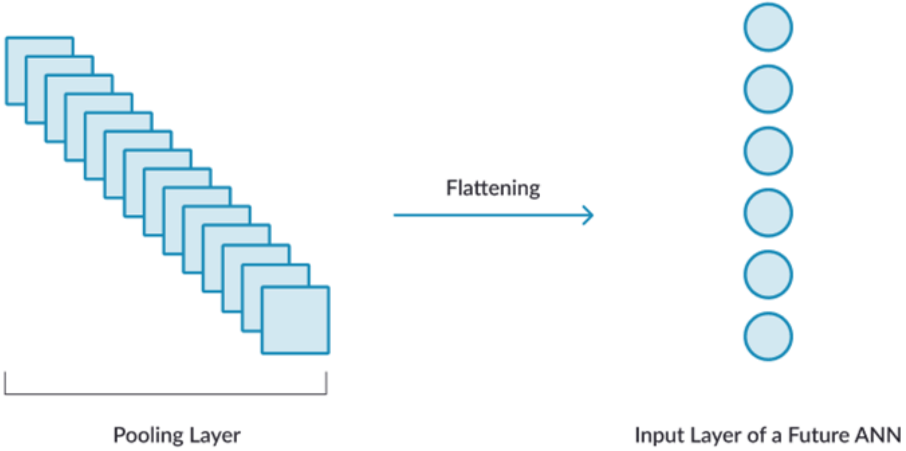
Flatten

Relu x2

Dropout

Softmax

03



ResNet50

Average Pooling 2D

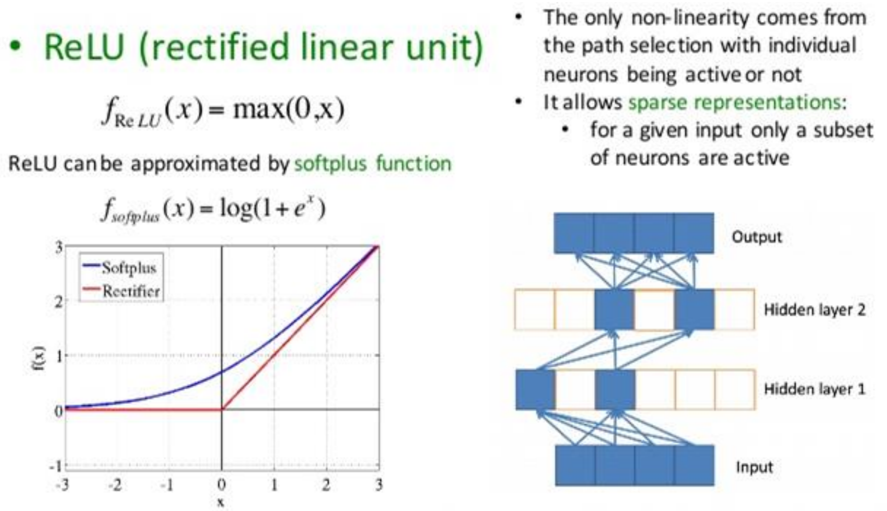
Flatten

Relu x2

Dropout

Softmax

03



- ReLU (rectified linear unit)

$$f_{\mathrm{Re}\,LU}(x) = \max(0,x)$$

ReLU can be approximated by softplus function

$$f_{softplus}(x) = \log(1+e^{x})$$

Softplus
Rectifier
f(x)
x
3
2
1
0
-1
-3
-2
-1
0
1
2
3
Output
Hidden layer 2
Hidden layer 1
Input

- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows sparse representations:
  - for a given input only a subset of neurons are active

ResNet50

Average Pooling 2D

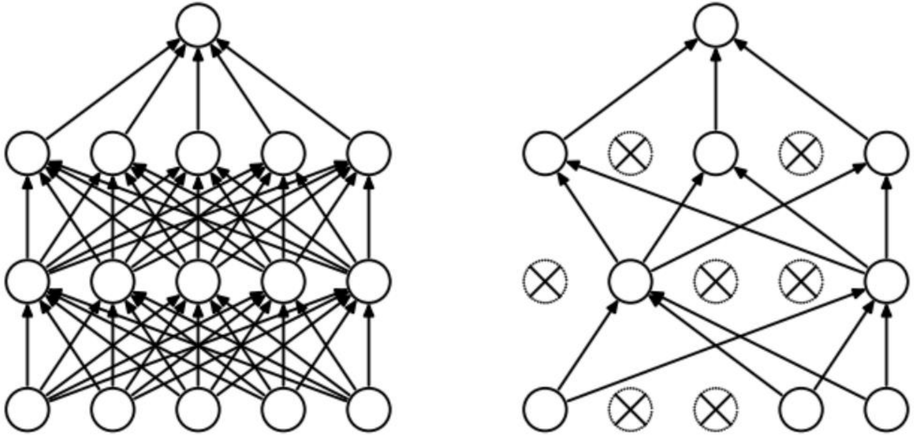
Flatten

Relu x2

Dropout

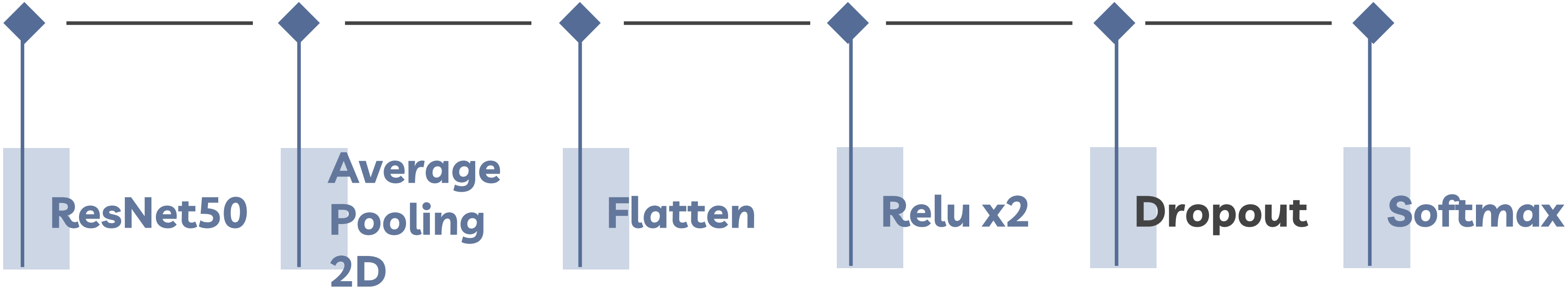
Softmax

03



(a) Standard Neural Net

(b) After applying dropout.



ResNet50

Average Pooling 2D

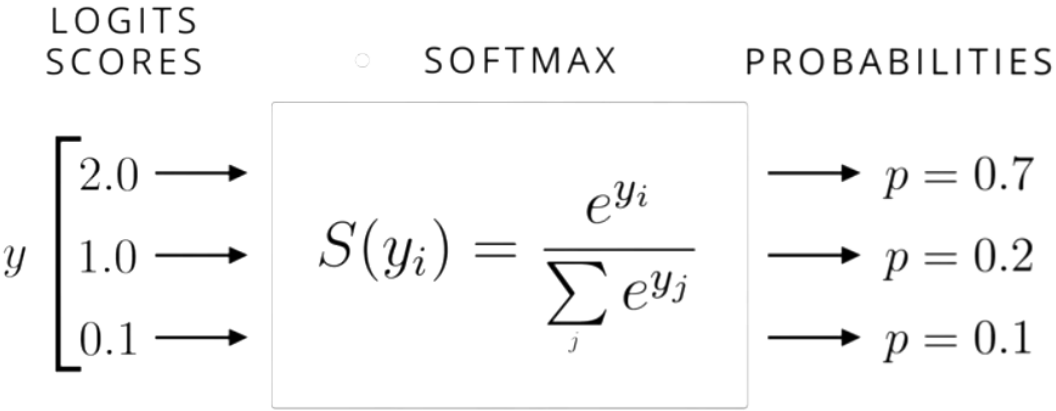
Flatten

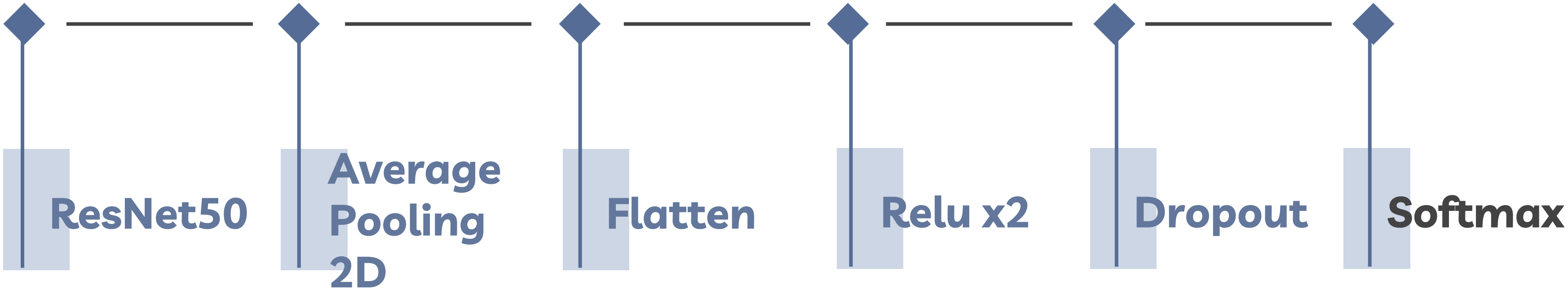
Relu x2

Dropout

Softmax

03





ResNet50

Average Pooling 2D

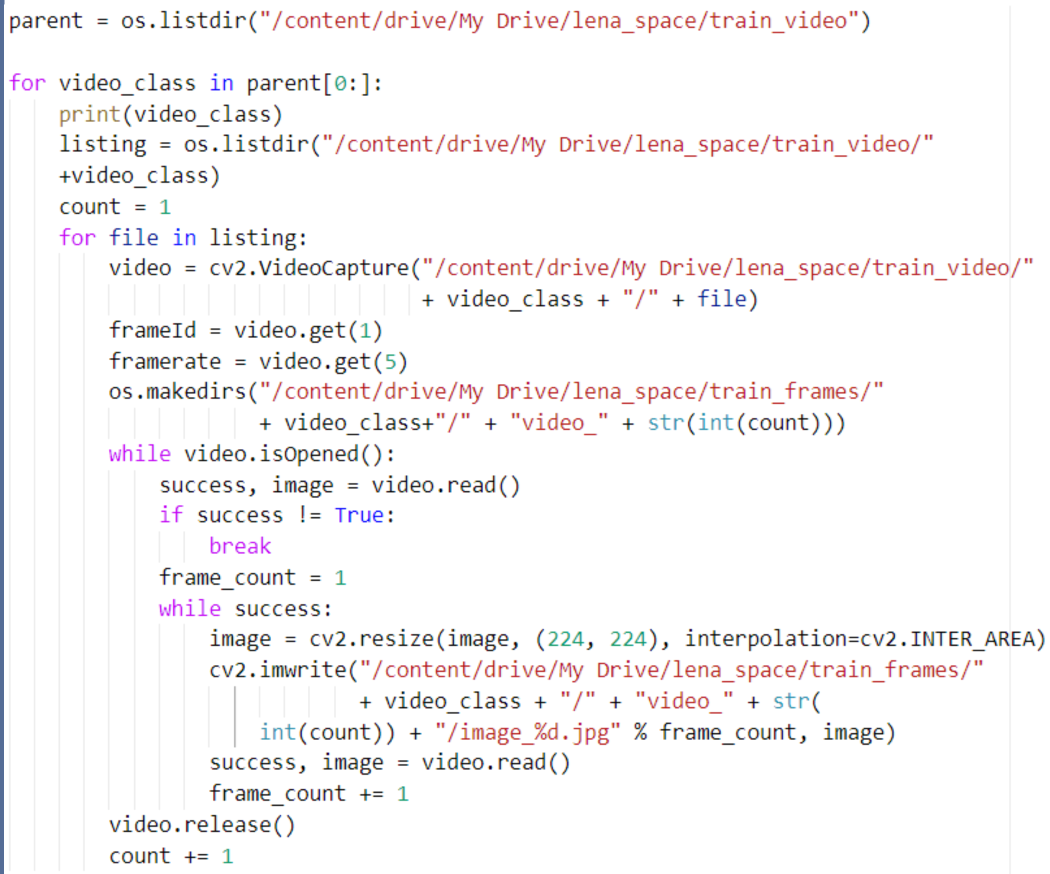
Flatten

Relu x2

Dropout

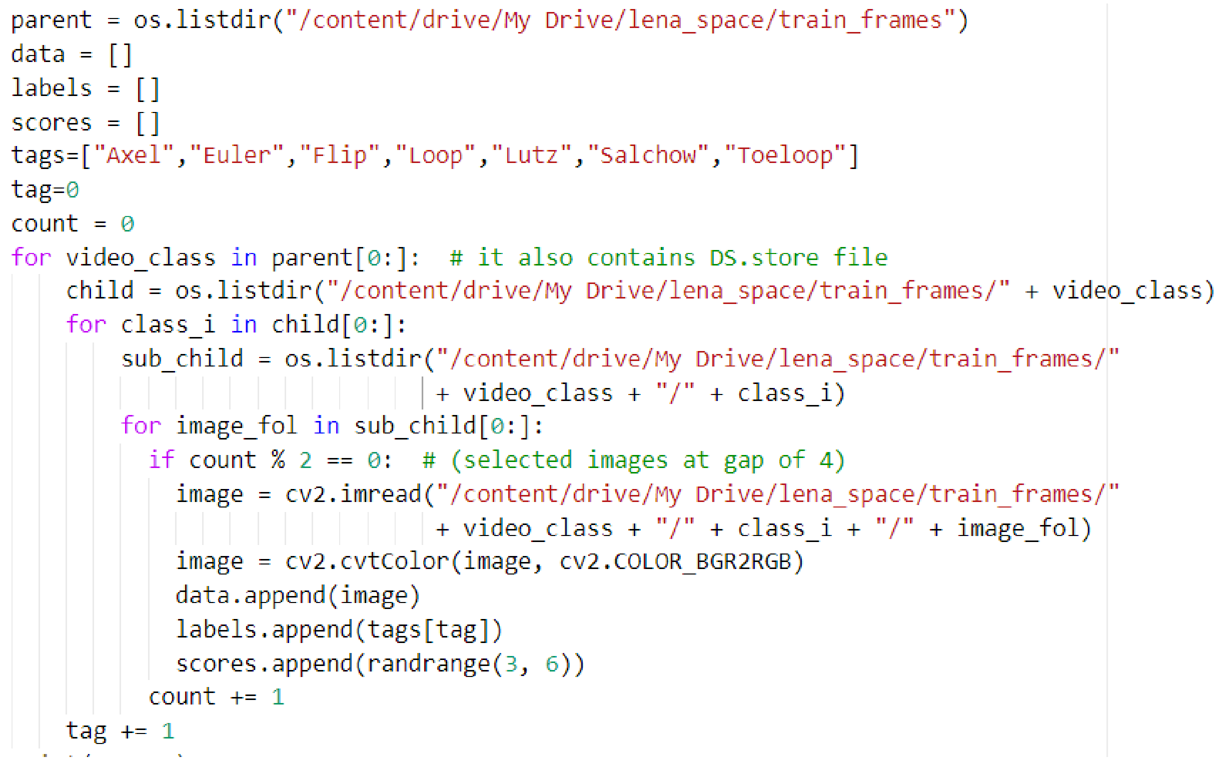
Softmax

04



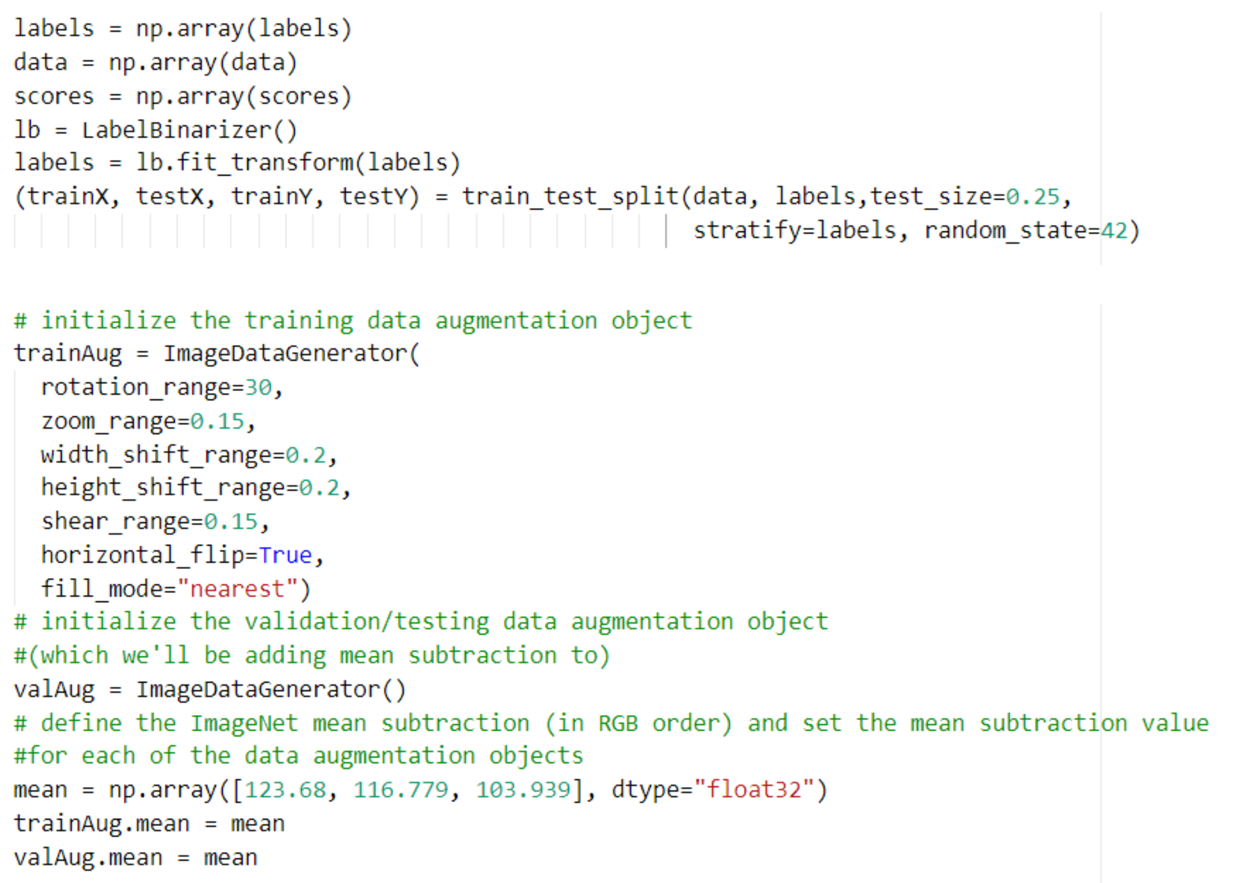
```
parent = os.listdir("/content/drive/My Drive/lena_space/train_video")

for video_class in parent[0:]:
    print(video_class)
    listing = os.listdir("/content/drive/My Drive/lena_space/train_video/"
    +video_class)
    count = 1
    for file in listing:
        video = cv2.VideoCapture("/content/drive/My Drive/lena_space/train_video/"
                                 + video_class + "/" + file)
        frameId = video.get(1)
        framerate = video.get(5)
        os.makedirs("/content/drive/My Drive/lena_space/train_frames/"
                    + video_class+"/" + "video_" + str(int(count)))
        while video.isOpened():
            success, image = video.read()
            if success != True:
                break
            frame_count = 1
            while success:
                image = cv2.resize(image, (224, 224), interpolation=cv2.INTER_AREA)
                cv2.imwrite("/content/drive/My Drive/lena_space/train_frames/"
                            + video_class + "/" + "video_" + str(
                    int(count)) + "/image_%d.jpg" % frame_count, image)
                success, image = video.read()
                frame_count += 1
        video.release()
        count += 1
```

04



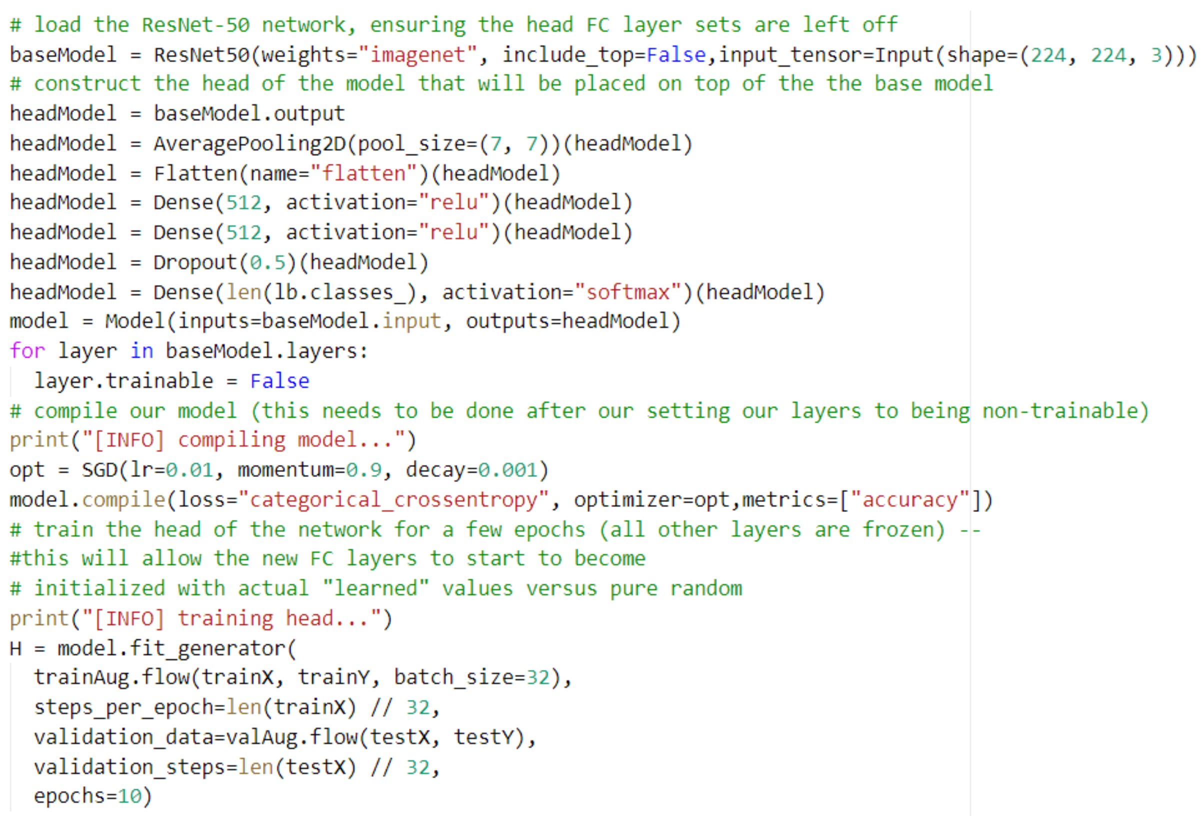
```
parent = os.listdir("/content/drive/My Drive/lena_space/train_frames")
data = []
labels = []
scores = []
tags=["Axel","Euler","Flip","Loop","Lutz","Salchow","Toeloop"]
tag=0
count = 0
for video_class in parent[0:]:  # it also contains DS.store file
    child = os.listdir("/content/drive/My Drive/lena_space/train_frames/" + video_class)
    for class_i in child[0:]:
        sub_child = os.listdir("/content/drive/My Drive/lena_space/train_frames/"
                               + video_class + "/" + class_i)
        for image_fol in sub_child[0:]:
          if count % 2 == 0:  # (selected images at gap of 4)
            image = cv2.imread("/content/drive/My Drive/lena_space/train_frames/"
                               + video_class + "/" + class_i + "/" + image_fol)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            data.append(image)
            labels.append(tags[tag])
            scores.append(randrange(3, 6))
          count += 1
    tag += 1
```

04



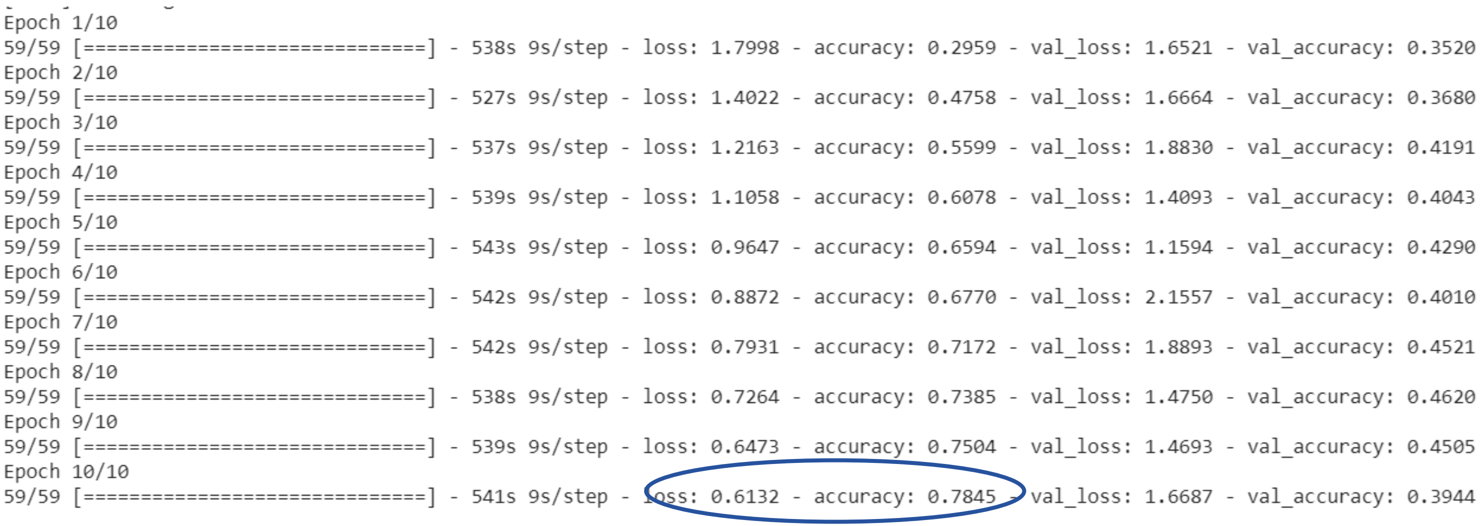
```
labels = np.array(labels)
data = np.array(data)
scores = np.array(scores)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
(trainX, testX, trainY, testY) = train_test_split(data, labels,test_size=0.25,
                         stratify=labels, random_state=42)


# initialize the training data augmentation object
trainAug = ImageDataGenerator(
 rotation_range=30,
 zoom_range=0.15,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.15,
 horizontal_flip=True,
 fill_mode="nearest")
# initialize the validation/testing data augmentation object
#(which we'll be adding mean subtraction to)
valAug = ImageDataGenerator()
# define the ImageNet mean subtraction (in RGB order) and set the mean subtraction value
#for each of the data augmentation objects
mean = np.array([123.68, 116.779, 103.939], dtype="float32")
trainAug.mean = mean
valAug.mean = mean
```

04



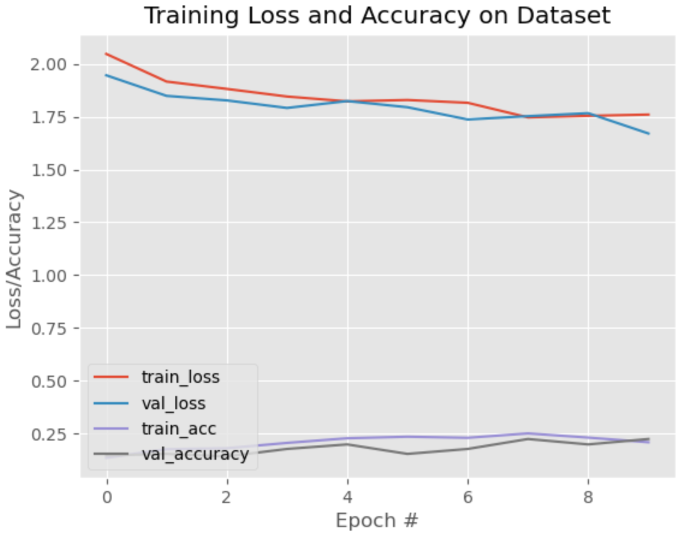
```
# load the ResNet-50 network, ensuring the head FC layer sets are left off
baseModel = ResNet50(weights="imagenet", include_top=False,input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(lb.classes_), activation="softmax")(headModel)
model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
# compile our model (this needs to be done after our setting our layers to being non-trainable)
print("[INFO] compiling model...")
opt = SGD(lr=0.01, momentum=0.9, decay=0.001)
model.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy"])
# train the head of the network for a few epochs (all other layers are frozen) --
#this will allow the new FC layers to start to become
# initialized with actual "learned" values versus pure random
print("[INFO] training head...")
H = model.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=32),
  steps_per_epoch=len(trainX) // 32,
  validation_data=valAug.flow(testX, testY),
  validation_steps=len(testX) // 32,
  epochs=10)
```

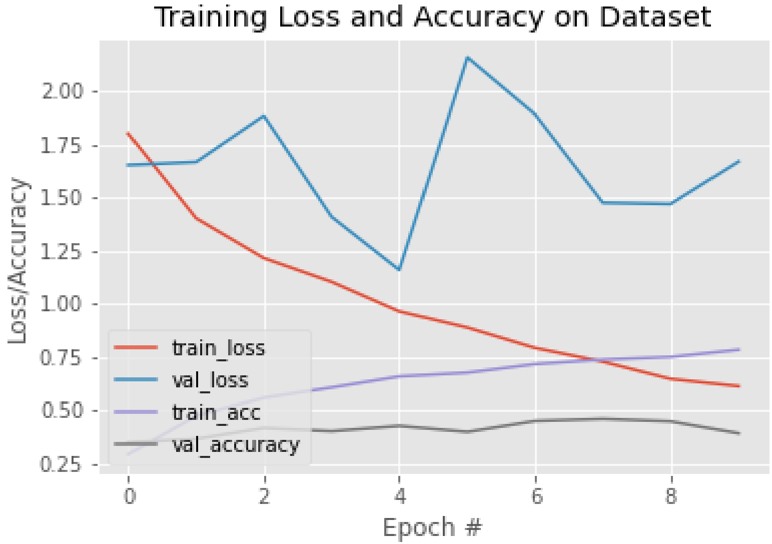
04



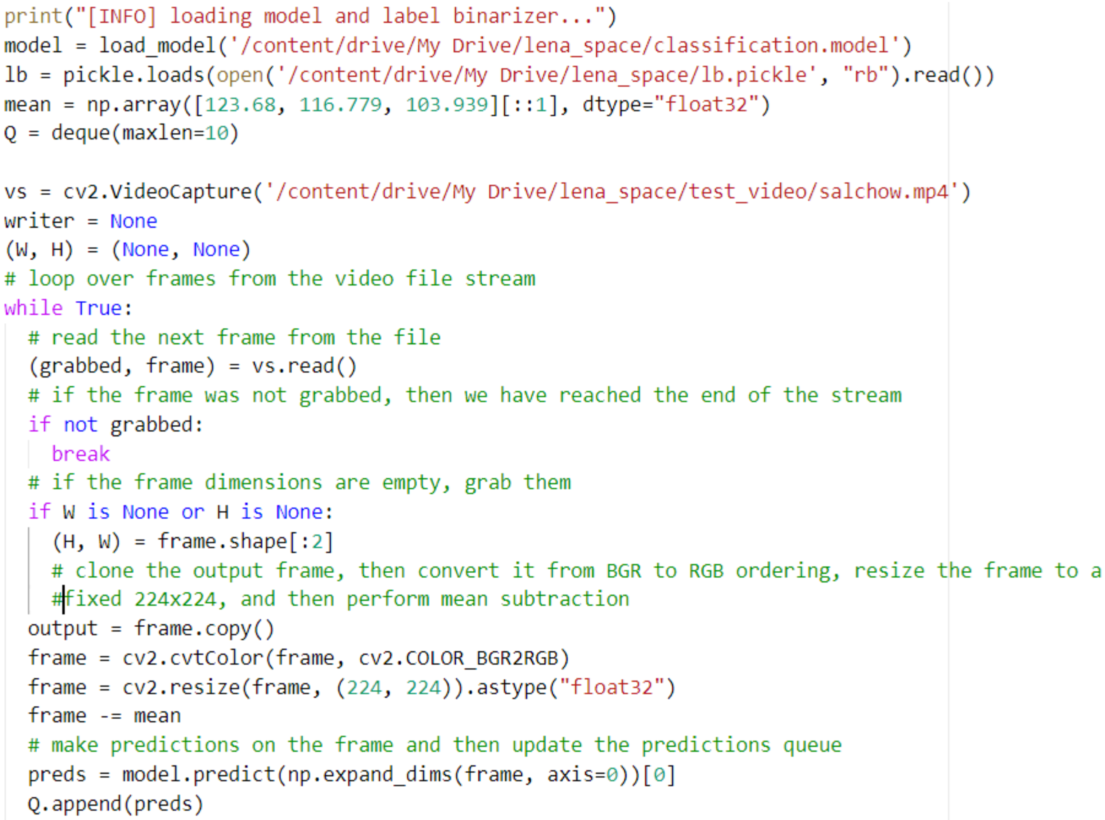
```
Epoch 1/10
59/59 [==============================] - 538s 9s/step - loss: 1.7998 - accuracy: 0.2959 - val_loss: 1.6521 - val_accuracy: 0.3520
Epoch 2/10
59/59 [==============================] - 527s 9s/step - loss: 1.4022 - accuracy: 0.4758 - val_loss: 1.6664 - val_accuracy: 0.3680
Epoch 3/10
59/59 [==============================] - 537s 9s/step - loss: 1.2163 - accuracy: 0.5599 - val_loss: 1.8830 - val_accuracy: 0.4191
Epoch 4/10
59/59 [==============================] - 539s 9s/step - loss: 1.1058 - accuracy: 0.6078 - val_loss: 1.4093 - val_accuracy: 0.4043
Epoch 5/10
59/59 [==============================] - 543s 9s/step - loss: 0.9647 - accuracy: 0.6594 - val_loss: 1.1594 - val_accuracy: 0.4290
Epoch 6/10
59/59 [==============================] - 542s 9s/step - loss: 0.8872 - accuracy: 0.6770 - val_loss: 2.1557 - val_accuracy: 0.4010
Epoch 7/10
59/59 [==============================] - 542s 9s/step - loss: 0.7931 - accuracy: 0.7172 - val_loss: 1.8893 - val_accuracy: 0.4521
Epoch 8/10
59/59 [==============================] - 538s 9s/step - loss: 0.7264 - accuracy: 0.7385 - val_loss: 1.4750 - val_accuracy: 0.4620
Epoch 9/10
59/59 [==============================] - 539s 9s/step - loss: 0.6473 - accuracy: 0.7504 - val_loss: 1.4693 - val_accuracy: 0.4505
Epoch 10/10
59/59 [==============================] - 541s 9s/step - loss: 0.6132 - accuracy: 0.7845 - val_loss: 1.6687 - val_accuracy: 0.3944
```

04


Training Loss and Accuracy on Dataset
Loss/Accuracy
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
2
4
6
8
Epoch #
train_loss
val_loss
train_acc
val_accuracy

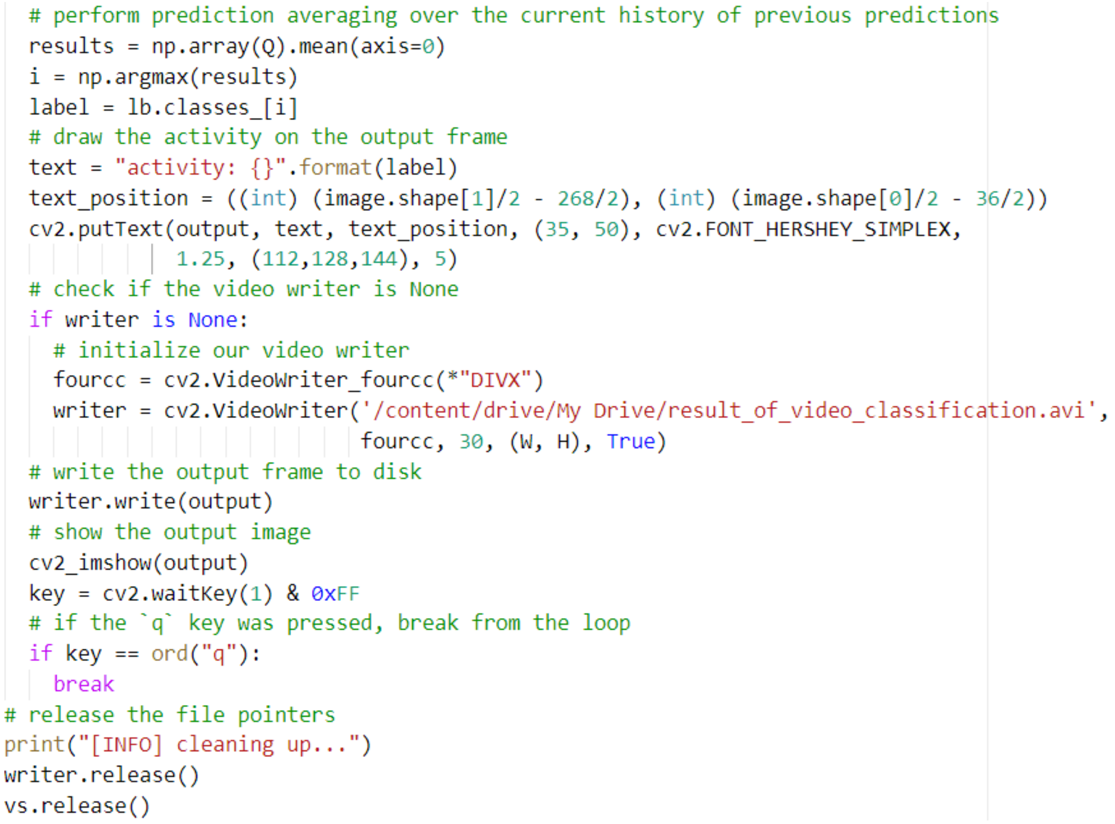

Training Loss and Accuracy on Dataset
Loss/Accuracy
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
2
4
6
8
Epoch #
train_loss
val_loss
train_acc
val_accuracy

04



```
print("[INFO] loading model and label binarizer...")
model = load_model('/content/drive/My Drive/lena_space/classification.model')
lb = pickle.loads(open('/content/drive/My Drive/lena_space/lb.pickle', "rb").read())
mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")
Q = deque(maxlen=10)

vs = cv2.VideoCapture('/content/drive/My Drive/lena_space/test_video/salchow.mp4')
writer = None
(W, H) = (None, None)
# loop over frames from the video file stream
while True:
  # read the next frame from the file
  (grabbed, frame) = vs.read()
  # if the frame was not grabbed, then we have reached the end of the stream
  if not grabbed:
    break
  # if the frame dimensions are empty, grab them
  if W is None or H is None:
    (H, W) = frame.shape[:2]
    # clone the output frame, then convert it from BGR to RGB ordering, resize the frame to a
    #fixed 224x224, and then perform mean subtraction
  output = frame.copy()
  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
  frame = cv2.resize(frame, (224, 224)).astype("float32")
  frame -= mean
  # make predictions on the frame and then update the predictions queue
  preds = model.predict(np.expand_dims(frame, axis=0))[0]
  Q.append(preds)
```

04



```
  # perform prediction averaging over the current history of previous predictions
  results = np.array(Q).mean(axis=0)
  i = np.argmax(results)
  label = lb.classes_[i]
  # draw the activity on the output frame
  text = "activity: {}".format(label)
  text_position = ((int) (image.shape[1]/2 - 268/2), (int) (image.shape[0]/2 - 36/2))
  cv2.putText(output, text, text_position, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,
              1.25, (112,128,144), 5)
  # check if the video writer is None
  if writer is None:
    # initialize our video writer
    fourcc = cv2.VideoWriter_fourcc(*"DIVX")
    writer = cv2.VideoWriter('/content/drive/My Drive/result_of_video_classification.avi',
                             fourcc, 30, (W, H), True)
  # write the output frame to disk
  writer.write(output)
  # show the output image
  cv2_imshow(output)
  key = cv2.waitKey(1) & 0xFF
  # if the `q` key was pressed, break from the loop
  if key == ord("q"):
    break
# release the file pointers
print("[INFO] cleaning up...")
writer.release()
vs.release()
```

05

activity: Axel
3 Loop
BV 4.90 GOE 0.14
C 12.08
WORLD FIGURE SK ING C AMPIONSHIPS
SAI MA RA 019
KATING CHAMPIONSHIPS
A JAPAN 2019
orldFigure
ISU
M マルハン M MARUHAN