NUS-ISS
Real Time Audio-Visual Sensing and Sense Making

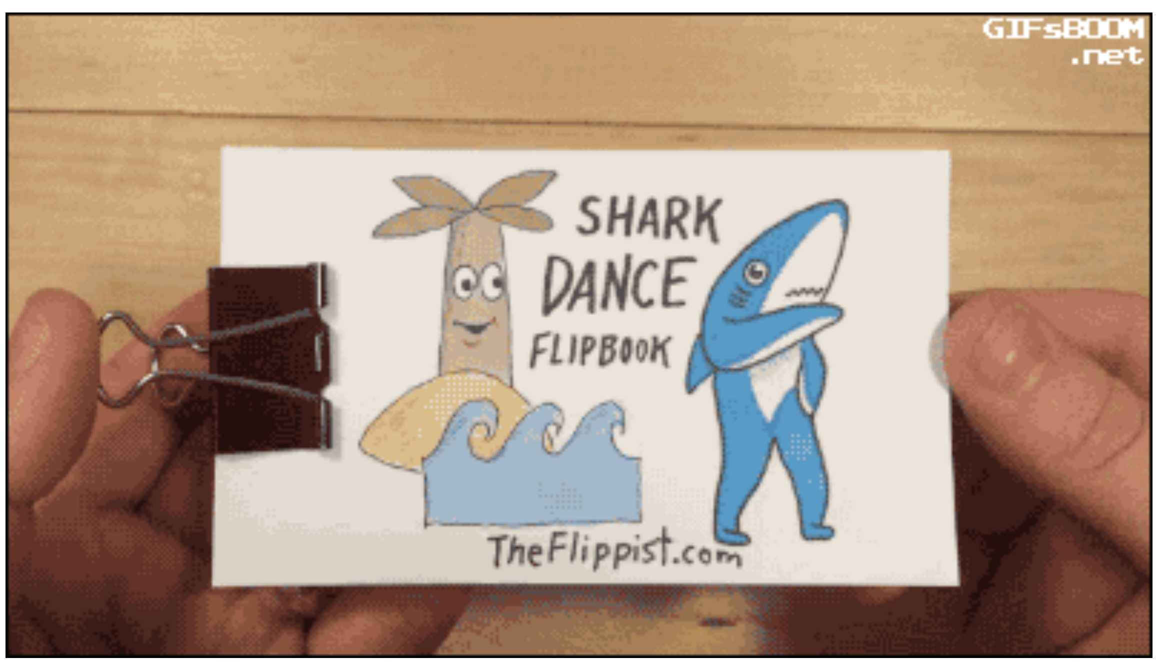
Module 8 - Sense making from multi-modal audio-visual data, part 1

Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

Video analysis using deep learning

Video analysis

Using deep learning



Source: TheFlippist.com

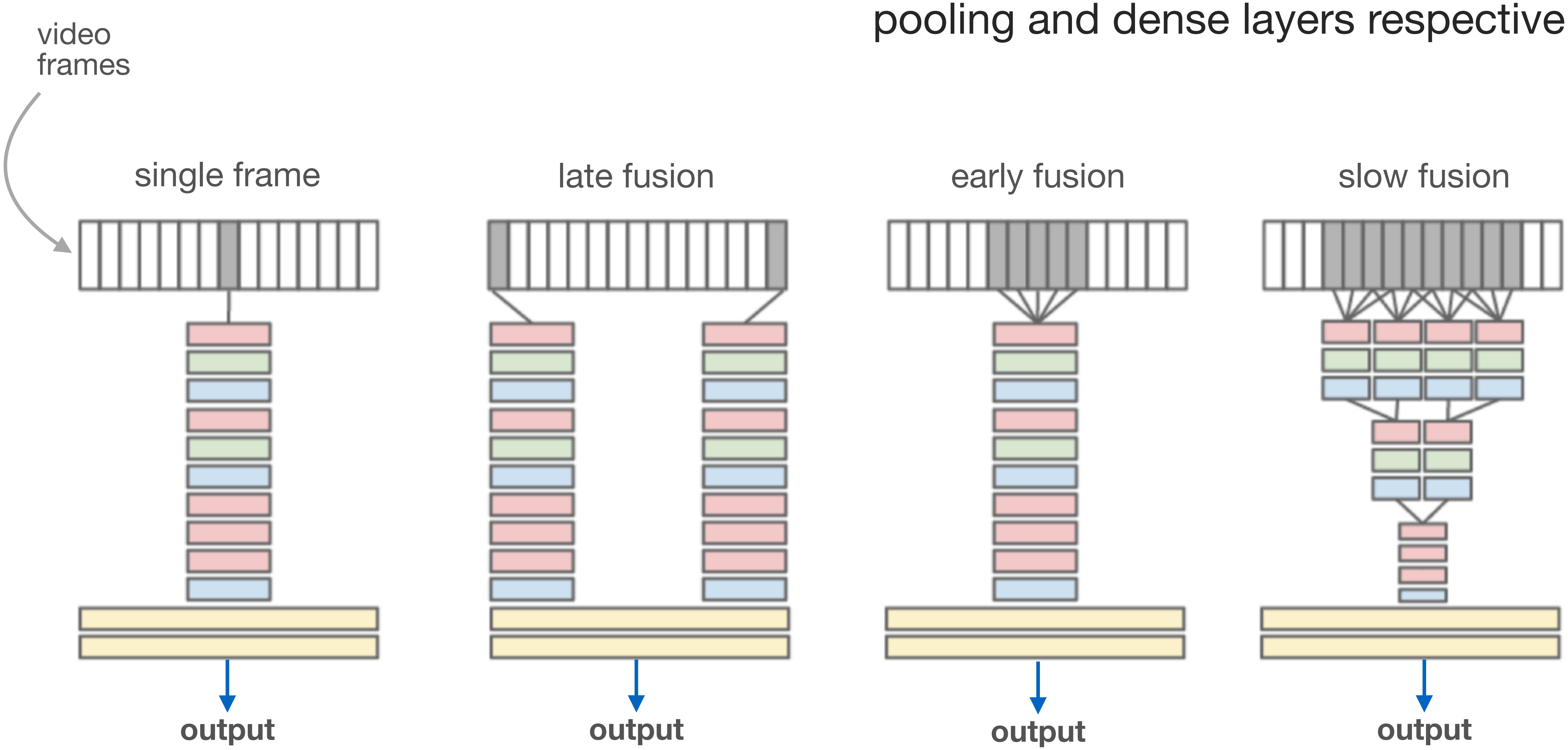
- Video: really just a stack of images ...

- In past, the standard approach to analyze video:
  1. Extract local visual features
  2. Combine features into video-level description
  3. Train a classifier (e.g. SVM) on the resulting "bag of words" representation

Video analysis

Using deep learning

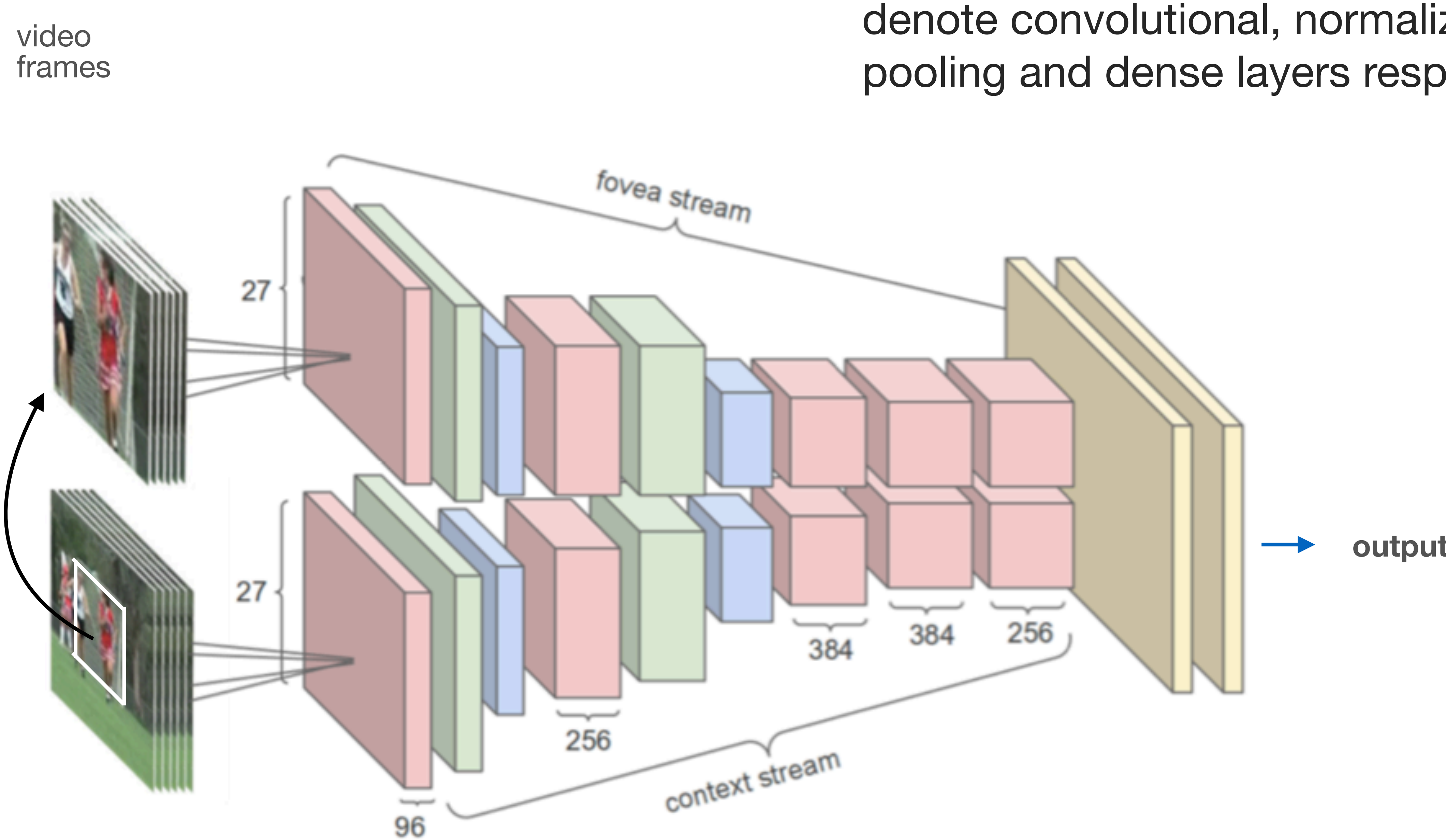
- With deep learning, few ways to analyze a video
- Red, green, blue and yellow boxes denote convolutional, normalization, pooling and dense layers respectively



Source: Large-scale video classification with convolutional neural network, by Karpathy et al.

Video analysis
Using deep learning

- Another possibility: two input frames, one the entire frame (context), another focus on the center of the frame (fovea)
- Red, green, blue and yellow boxes denote convolutional, normalization, pooling and dense layers respectively



Source: Large-scale video classification with convolutional neural network, by Karpathy et al.

Video analysis
Using deep learning

- The simplest way to use deep learning to analyze video: grab frame by frame, feed single frame to deep learning model, then get output

- Simple and effective, but with a problem: unstable output

Video analysis

Using deep learning

- Why is this happening?
  - Because we analyze video frame by frame
  - Performance of deep learning model is not perfect, easily swayed by unseen objects or events in image

- Can we get rid of this? Or more importantly, is it possible to use the analysis from the previous frames to make better judgement on the current frame without using early, late or slow fusion?

Let's look at one possible solution

Event identification
in video

- Identify 3 types of events in video: volleyball, badminton, formula 1
- Go to this github (https://github.com/anubhavmaity/), find the link to download the relevant images

Transfer learning

To leverage ....

Source: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

- A common and highly effective approach on small image datasets

- Use a pre-trained network that was previously trained on a large-scale dataset

- If the original dataset is large and general enough, the pre-trained model may have actually learned how to extract the ***generic*** features (e.g. edges, lines, shapes, textures and etc.) that truly matters to vision

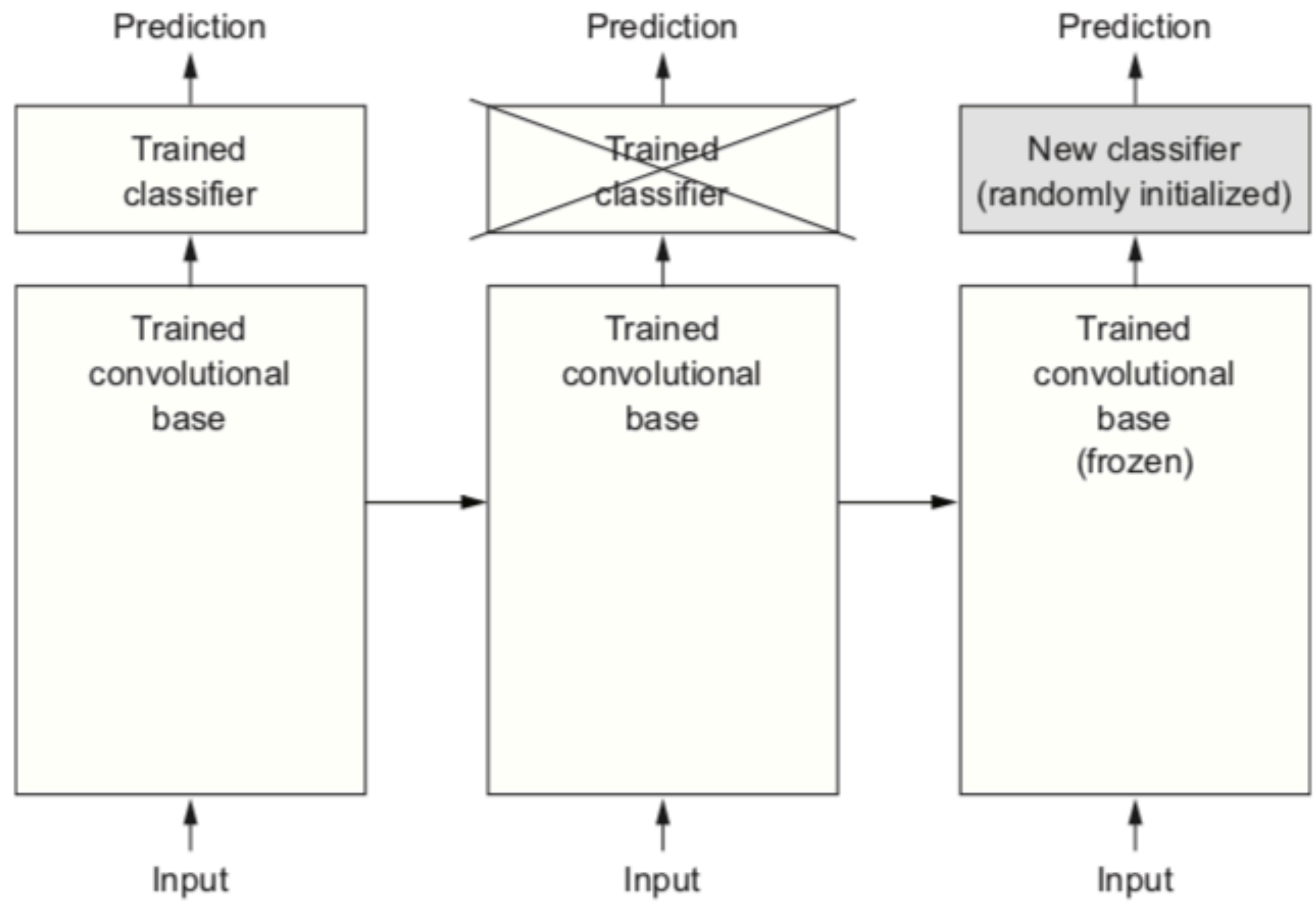
- Thus, the way the net extracts features is useful to many different vision problem

Source: Deep learning with Python by Francois Chollet

Transfer learning

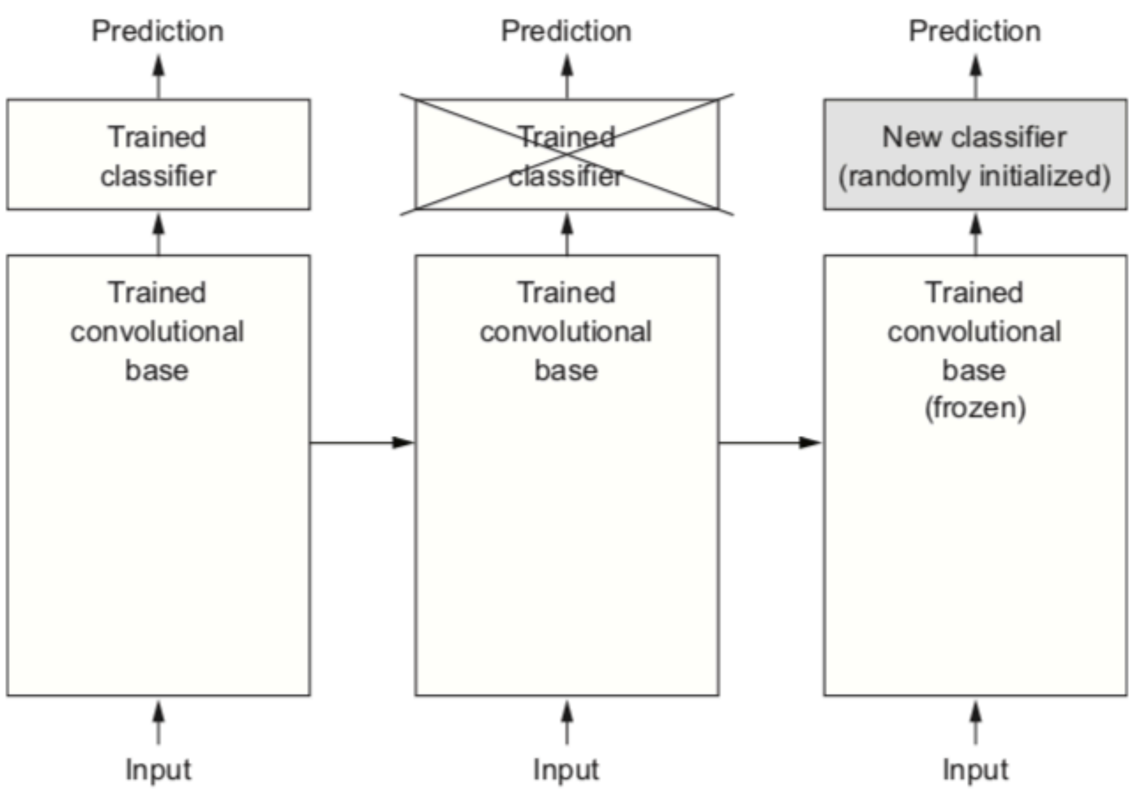
Feature extraction

- Two approaches to use pre-trained network: feature extraction and fine-tuning



Source: Deep learning with Python by Francois Chollet

Transfer learning
Feature extraction



Source: Deep learning with Python by Francois Chollet

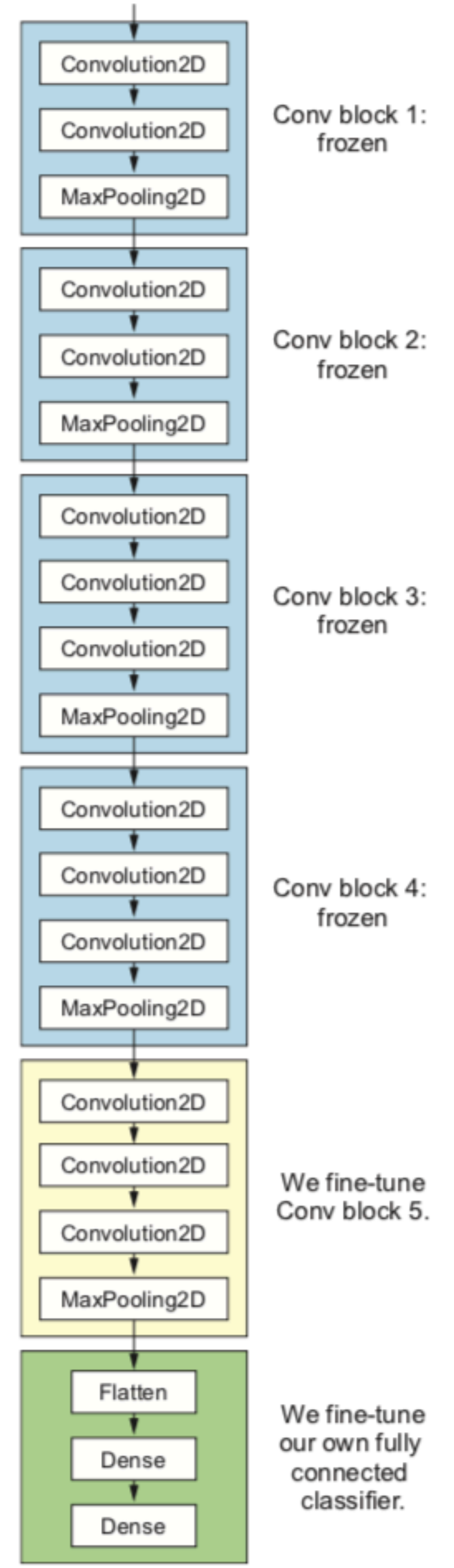
- Can we re-use the classifier?
- Representations learned by convnet base likely more generic, thus reusable
- Representations learned by classifier more specific to the set of classes the model was arranged to be trained on, not so reusable

Transfer learning

Fine tuning

- **After we have trained the added classifier with the convnet base frozen**, we can unfreeze a few layers of the frozen base model that are near to the classifier
- Fine-tuning: we train the newly unfreezed layers together with the classifier

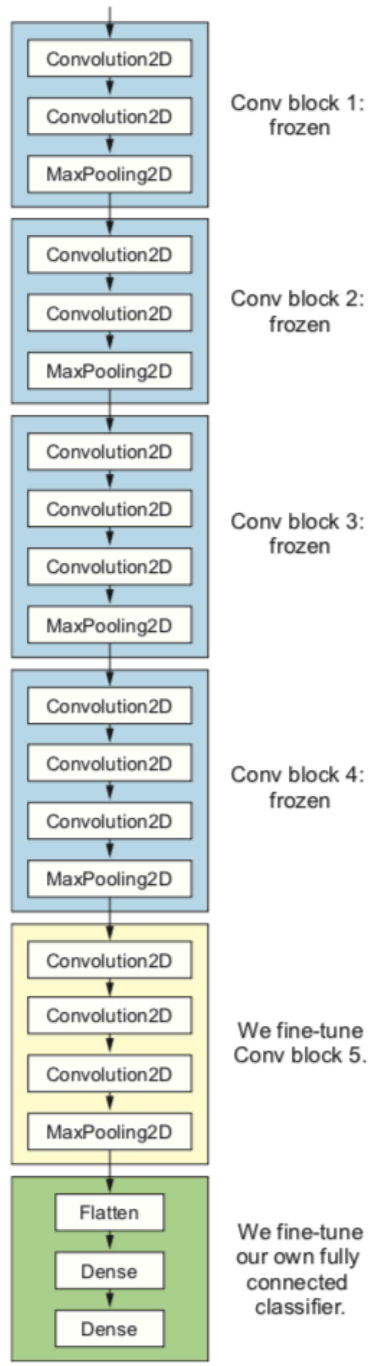
VGG16



Source: Deep learning with Python by Francois Chollet

Transfer learning

Fine tuning



Source: Deep learning with Python by Francois Chollet

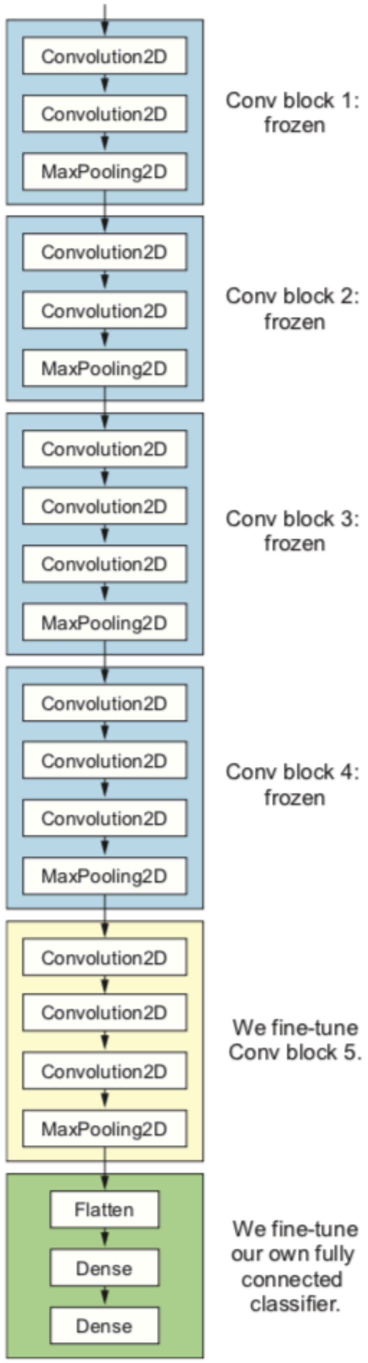
- Must the classifier be trained before we perform a fine-tuning?

- Yes, else the errors back-propagated will be too large and distort the weights of the just unfreezed layers

- Note: we are doing **fine-tuning** on both the unfreezed layers and classifier, **not re-train** the both

Transfer learning

Fine tuning



Source: Deep learning with Python by Francois Chollet

- Can unfreeze and fine-tune more layers?

- Things to consider: Earlier layers in the base net (layers that are near to input) encode more generic and more reusable features; this is something you want to keep

- Later layers are more specific, you want to fine-tune them to make them fit for your problem

- Pre-trained networks usually have much more parameters and are much more powerful (that's why you want to use them); re-train more layers will easily lead to overfitting on small dataset

Import the necessary

Quite a few things to import

```
> from tensorflow.keras.preprocessing.image import ImageDataGenerator
> from tensorflow.keras.layers import AveragePooling2D
> from tensorflow.keras.applications import ResNet50
```

the convnet base we are going to use

```
> from tensorflow.keras.layers import Dropout
> from tensorflow.keras.layers import Flatten
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Input
> from tensorflow.keras.models import Model
> from tensorflow.keras.optimizers import SGD
> from tensorflow.keras.callbacks import ModelCheckpoint,CSVLogger

> from sklearn.preprocessing import LabelBinarizer
> from sklearn.model_selection import train_test_split

> from imutils import paths
```

The function from this library can help to get all the images in a folder, instead of we search through file extension

```
> import matplotlib.pyplot as plt
> import numpy as np
> import pickle
```

this is used to save the fitted label binarizer

```
> import cv2
> import os
> import sklearn.metrics as metrics
```

Some basic setup

```
> labels    = ["badminton", "volleyball", "formula1"]
> fdr       = 'data'
```

The folder that stores all the images

```
> imgMean   = np.array([123.68,116.779,103.939],
                        dtype="float32")
```

The mean to be subtracted from each image

```
> plt.style.use('ggplot')
> plt.rcParams['ytick.right']     = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']      = False
> plt.rcParams['ytick.labelleft'] = False
> plt.rcParams['font.family']     = 'Arial'
```

Setting up the style of our plot

Load the images and labels

Use 'paths' from 'imutil' library, it can easily get all the images in this folder (including subfolder) without the need to specifying file extension

```
> imgPaths          = list(paths.list_images(fdr))
```

```
imgPaths | list   | 2307    | ['data/volleyball/00000428.jpg','data ...
```

```
> dat               = []
> lbl               = []


> for pth in imgPaths:
      l             = pth.split(os.path.sep)[-2]
      img           = cv2.imread(pth)
      img           = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
      img           = cv2.resize(img, (224, 224))

      dat.append(img)
      lbl.append(l)
```

Get the label from the path

Read the image

Remember to convert BGR to RGB

Resize the image, (224, 224) is the required input size to ResNet50

dat	list	2307	[Numpy array, Numpy array, Numpy array, Numpy array, Numpy array, Nump ...
lbl	list	2307	['volleyball', 'volleyball', 'volleyball', 'volleyball', 'volleyball', ...

Load the images and labels

- How actually we get the label from the path

```
> spth        = imgPaths[1]
> spth
: 'data/volleyball/00000366.jpg'
```

Get the second item from imgPaths

```
> op          = spth.split(os.path.sep)
```

Split the string according to the OS path separator (I am using Mac while working on this, that's why it will be '/')

```
> op
: ['data', 'volleyball', '00000366.jpg']
```

```
> op[-1]
: '00000366.jpg'
```

The last item

```
> op[-2]
: 'volleyball'
```

The second last item

Prepare data and labels

- LabelEncoder: Encode each label to a single integer
- LabelBinarizer: One-hot encode each label

```
> dat                = np.array(dat)
> lbl                = np.array(lbl)
```

Convert 'dat' and 'lbl' from list to numpy array

```
> lb                 = LabelBinarizer()
> lbl                = lb.fit_transform(lbl)
```

One-hot encode the labels

dat	uint8	(2307, 224, 224, 3)	[[[[186 158 157] [201 158 158]
lbl	int64	(2307, 3)	[[0 0 1] [0 0 1]

```
> (trDat,
  tsDat,
  trLbl,
  tsLbl)             = train_test_split(dat,
                                        lbl,
                                        test_size=0.25,
                                        stratify=lbl,
                                        random_state=331)
```

Prepare data and labels

- Image augmentation through image data generator

```
> trDatGen         = ImageDataGenerator(rotation_range=30,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        zoom_range=0.15,
                                        shear_range=0.15,
                                        horizontal_flip=True,    only horizontal flip, no vertical
                                        fill_mode="nearest")

> tsDatGen         = ImageDataGenerator()

> trDatGen.mean    = imgMean
> tsDatGen.mean    = imgMean
```

Setup model

```
> optmz   = SGD(lr=1e-4,
                momentum=0.9,
                decay=1e-4/25)
> base    = ResNet50(weights="imagenet",
                     include_top=False,
                     input_tensor=Input(shape=(224, 224, 3)))
> def createModel():
      h   = base.output
      h   = AveragePooling2D(pool_size=(7, 7))(h)
      h   = Flatten(name="flatten")(h)
      h   = Dense(512, activation="relu")(h)
      h   = Dropout(0.5)(h)
      h   = Dense(len(lb.classes_), activation="softmax")(h)
      model = Model(inputs=base.input, outputs=h)
      for layer in base.layers:
          layer.trainable     = False
      model.compile(loss="categorical_crossentropy",
                    optimizer=optmz,
                    metrics=["accuracy"])
      return model
```

False to NOT include the dense layers (classifer)

Set 'False' not to train the base

The decay ....

How does it work in Keras

- When we set decay in optimizer, Keras adjust the learning rate after ***every batch update***
- Each batch update is also called an iteration
- If a training dataset has 50,000 images batch size is set to 20, then we have 2,500 batch updates or iterations in each epoch

$$\mathrm{LR} = \mathrm{LR}_{\mathrm{init}} \times \frac{1.0}{1.0 + \mathrm{decay} \times \mathrm{iterations}}$$

Almost there ..

A few things to do before training

```
> model         = createModel()
> modelGo       = createModel()
```

for training

for testing

```
> model.summary()
> modelname        = 'sportsV1'
> modelpath        = os.path.join('model',modelname+".hdf5")
> checkpoint       = ModelCheckpoint(modelpath,
                                     monitor='val_loss',
                                     verbose=0,
                                     save_best_only=True,
                                     mode='min')
```

save the model in 'model' folder

save the model with the mininum val_loss, not maximum validation accuracy

```
> csv_logger       = CSVLogger(modelname +'.csv')
> callbacks_list   = [checkpoint,csv_logger]
```

Training

Before we train, we need to save an important object

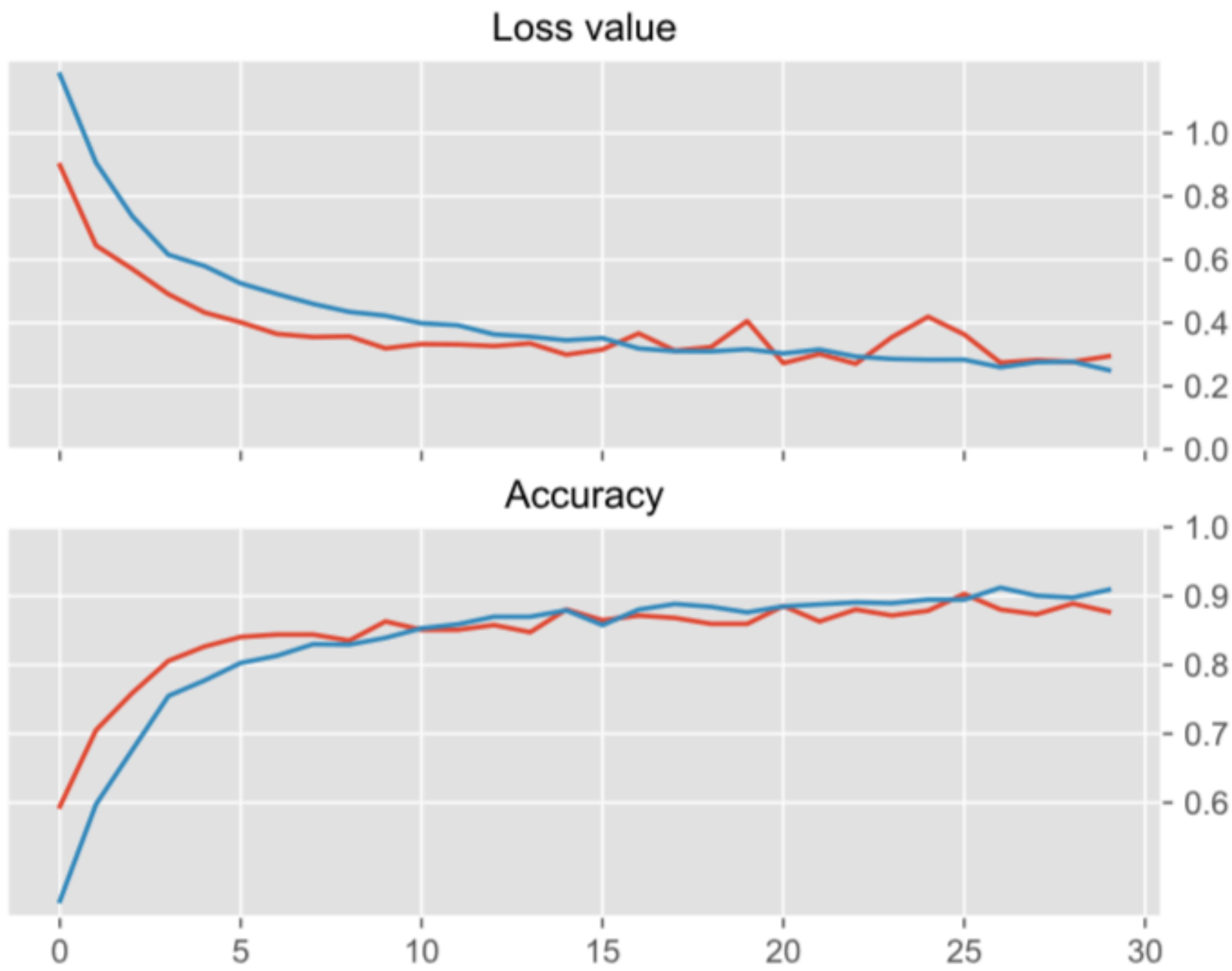
- Need to save LabelBinarizer object so that in future we know how to decode the output from network
- Use pickle to serialize python object into a character stream

```
> pickpath        = os.path.join('model',modelname+".pickle")
> f               = open(pickpath,"wb")
> f.write(pickle.dumps(lb))
> f.close()
```

save the object in 'model' folder

```
> model.fit_generator(trDatGen.flow(trDat, trLbl, batch_size=32),
                      steps_per_epoch=len(trDat)//32,
                      validation_data=tsDatGen.flow(tsDat, tsLbl),
                      validation_steps=len(tsLbl)//32,
                      epochs=30,
                      callbacks=callbacks_list)
```

Training
Result



training
testing

Training
Result

```
Best accuracy (on testing dataset): 88.04%
              precision    recall  f1-score   support

   badminton     0.8000    0.9655    0.8750       232
    formula1     0.9806    0.8994    0.9383       169
  volleyball     0.9296    0.7500    0.8302       176

    accuracy                         0.8804       577
```

Confusion matrix:

```
badminton   [[224   3   5]
formula1     [ 12 152   5]
volleyball   [ 44   0 132]]
```

Let's create the code to analyze video

Import the necessary

Quite a few things to import

```
> from tensorflow.keras.layers import AveragePooling2D
> from tensorflow.keras.applications import ResNet50
> from tensorflow.keras.layers import Dropout
> from tensorflow.keras.layers import Flatten
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Input
> from tensorflow.keras.models import Model
> from tensorflow.keras.optimizers import SGD
> from collections import deque
```

This is the key to get stable prediction output

```
> import numpy as np
> import pickle
> import cv2
> import os
```

Basic setup

```
> labels        = ["badminton", "volleyball", "formula1"]
> qsize         = 32
> videoName     = 'formula2.mp4'
> outName       = videoName[:-4]+'_'+str(qsize)+'.avi'
> modelname     = 'sportsV1'

> modelpath     = os.path.join('model',modelname+".hdf5")
> pickpath      = os.path.join('model',modelname+".pickle")
> videopath     = os.path.join('example_clips',videoName)
> outpath       = os.path.join('output',outName)

> imgMean       = np.array([123.68,116.779,103.939],
                       dtype="float32")
> Q             = deque(maxlen=qsize)
```

The size for the deque object

The video to be analyzed

The filename of the output

The model to be loaded

The path to the model

The path to the pickle file

The path to the video

The path to the output

The mean to be subtracted for input to resnet50

You can imagine the deque object as a list, in this case the size of the 'list' is restricted to be 32 and with a first-in first-out property

Example on deque

```
> from collections import deque

> four_numbers = deque([2, 3, 4], maxlen=4)
> four_numbers
out: deque([2, 3, 4])

> four_numbers.append(5)
> four_numbers
out: deque([2, 3, 4, 5])

> four_numbers.append(6)
> four_numbers
out: deque([3, 4, 5, 6])                    item ‘2’ is kicked out

> four_numbers.append(7)
> four_numbers
out: deque([4, 5, 6, 7])                    item ‘3’ is kicked out
```

Load model

Define model

```
> lb          = pickle.loads(open(pickpath, "rb").read())
> optmz       = SGD(lr=1e-4,
                    momentum=0.9,
                    decay=1e-4/25)
> base        = ResNet50(weights=None,
                         include_top=False,
                         input_tensor=Input(shape=(224, 224, 3)))
> def createModel():
      h   = base.output
      h   = AveragePooling2D(pool_size=(7, 7))(h)
      h   = Flatten(name="flatten")(h)
      h   = Dense(512, activation="relu")(h)
      h   = Dropout(0.5)(h)
      h   = Dense(len(lb.classes_), activation="softmax")(h)

      model = Model(inputs=base.input, outputs=h)

      for layer in base.layers:
          layer.trainable     = False

      return model
```

Load the pickle file

There is no need to load 'imagenet' weight since we are going to load our own weights

The definition of the model must be the same with the one we defined in the training

Load model

Load weights

- Use model.load_weights to load the weights

```
> model       = createModel()
> model.load_weights(modelpath)
> model.compile(loss='categorical_crossentropy',
                optimizer=optmz,
                metrics=['accuracy'])
```

Reading video

cv2.VideoCapture

- In opencv, use VideoCapture object to read a video
- The input can be a file, or integer specify camera
- If there is only one camera connected to computer, set 0
- If there are two cameras, to get the stream from second camera, set 1

```
vs    = cv2.VideoCapture('testvideo.avi')
vs    = cv2.VideoCapture(0)
vs    = cv2.VideoCapture(1)
```

This line reads in a video file

This line reads in a video stream from the camera connected to computer

This line reads in a video stream from the second camera connected to computer (if there are two cameras connected to computer)

Reading video

cv2.VideoCapture

- After setting up VideoCapture object, we use `.read()` to read the next frame in video stream
- It gives two output, one is boolean, another is the image

```
> vs              = cv2.VideoCapture('formula2.mp4')
> (grabbed,frame1) = vs.read()       load the first frame
> (grabbed,frame2) = vs.read()       load the second frame
```

```
> plt.imshow(frame1)
> plt.axis('off')
```


```
> plt.imshow(frame2)
> plt.axis('off')
```

Analyze video

in a while loop

- To analyze video, we loop through each frame under a while loop
- We break the loop when there is no more frame to read, in this case the `grabbed` will be `False`

```
> vs       = cv2.VideoCapture(videopath)
> writer   = None
> (W, H)   = (None, None)

> while True:
      (grabbed,
       frame)     = vs.read()

      if not grabbed:
          break

      if W is None or H is None:
          (H, W)  = frame.shape[:2]

      ...
```

this is for writing video

To store width and height of frame

Read the next frame

When there is no more frame to read, break the while loop

Get the frame size, take note, H is height which is the number of rows in an array, and W is width, which is the number of columns in an array

Analyze video

in a while loop

- Make classification

```
> while True:
      ...

      output  = frame.copy()
      frame   = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
      frame   = cv2.resize(frame, (224, 224)).astype("float32")
      frame  -= imgMean

      preds   = model.predict(np.expand_dims(frame,axis=0))[0]

      Q.append(preds)
      predout = np.array(Q).mean(axis=0)

      clss    = np.argmax(predout)
      label   = lb.classes_[clss]

      ...
```

Make a copy of the 'frame' array

Convert the image from BGR to RGB

Resize and convert to float32

Subtract the image from the mean

We turn the frame from (224,224,3) to (1,224,224,3) and feed the input to model and make prediction

The output from model.predict is 2D array, take the first row and get a 1D array

Append the prediction to 'Q', the deque object and get the mean on the past 32 prediction

Get the class with the maximum average probability and get its corresponding text from the LabelBinarizer

Analyze video

in a while loop

- Put the text to each frame and save the frame to video

```
> while True:
      ...

      text        = "Event: {}".format(label)
      cv2.putText(output,
                  text,
                  (10,40),
                  cv2.FONT_HERSHEY_SIMPLEX,
                  1.25,
                  (0,255,0),
                  5,
                  cv2.LINE_AA)
      if writer is None:
          fourcc = cv2.VideoWriter_fourcc(*"MJPG")
          writer = cv2.VideoWriter(outpath,
                                   fourcc,
                                   30,
                                   (W, H),
                                   True)
      writer.write(output)
```

The text to put on each frame

Put the text on each image

If the writer is not setup, setup now

Specify the codec we want to use

Path to output

The codec to be used

The frame rate

The frame size

If the video is colour

write the frame to video

Closing

two more things to do

- We need to release video writer and video capture after all these end

```
> writer.release()
> vs.release()
```

fourcc
fourcc.org

- fourcc stands for "four character code", an identifier for a video codec, compression format, colour or pixel format used in media files

- 'MJPG' stands for Motion JPEG codec

- Check out for more at www.fourcc.org