Introduction to

https://keras.io/

What is keras?

“Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.”

Keras program structure

./2_keras_intro/examples/e1_Sequential_keras.py

```python
import keras
from keras.datasets import cifar10
from keras import layers
from sklearn import cross_validation
from sklearn.utils import shuffle
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import OneHotEncoder

N_EPOCHS= 20
BATCH_SIZE= 32
LEARNING_RATE= 1e-2  #PLAY with learning rate. try 1e-1, 1e-2 ...
N_HIDDEN= 64

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
#Rescale data.
x_train= x_train /255.0
x_test=  x_test  /255.0
print(x_train.shape, y_train.shape, x_train.max(), x_train.min() )
#One-hot-encode labels
oh_encoder= OneHotEncoder(n_values=10, sparse=False)
y_train= oh_encoder.fit_transform(y_train)
y_test= oh_encoder.transform(y_test)
print(x_train.shape, y_train.shape)

print("data ready")
#Model definition

model = keras.models.Sequential()
model.add( layers.Conv2D(filters=32, kernel_size= 5, strides=1,padding='same',
                         activation='relu',  input_shape= x_train.shape[1:] ))
model.add( layers.MaxPooling2D(pool_size=4, strides=2 ) )
model.add( layers.Flatten() )
model.add( layers.Dense(N_HIDDEN, activation='relu'))
model.add( layers.Dense(y_train.shape[-1], activation='softmax'))
sgd= keras.optimizers.SGD(lr=LEARNING_RATE, decay=0., momentum=0., nesterov=False)
#Model compilation or loading
model.compile(loss='categorical_crossentropy',optimizer=sgd, metrics=['accuracy'])
print("network created")
print("training begin")
#Model training
model.fit(x_train, y_train,epochs=N_EPOCHS, batch_size=BATCH_SIZE, validation_split=0.1)
#Model evaluation
score = tuple(model.evaluate(x_test, y_test, batch_size=BATCH_SIZE))
print("Testing evaluation loss %f acc %f roc_auc %f"%score)
```

Data preparation

Model definition

training

evaluation

Data preparation

- If data fits in memory, load it as numpy arrays and train, evaluate or use your model with model.fit(), model.evaluate(), model.predict().

```
14 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
15 #Rescale data.
16 x_train= x_train /255.0
17 x_test=  x_test  /255.0
18 print(x_train.shape, y_train.shape, x_train.max(), x_train.min() )
19 #One-hot-encode labels
20 oh_encoder= OneHotEncoder(n_values=10, sparse=False)
21 y_train= oh_encoder.fit_transform(y_train)
22 y_test= oh_encoder.transform(y_test)
23 print(x_train.shape, y_train.shape)
24
25 print("data ready")
```

```
40 #Model training
41 model.fit(x_train, y_train,epochs=N_EPOCHS, batch_size=BATCH_SIZE, validation_split=0.1)
42 #Model evaluation
43 score = tuple(model.evaluate(x_test, y_test, batch_size=BATCH_SIZE))
44 print("Testing evaluation loss %f acc %f roc_auc %f"%score)
```

- Otherwise, define a python generator that produces batches or a keras.utils.Sequence class and train, evaluate or use your model with model.fit_generator(), model.evaluate_generator(), model.predict_generator().
- If you want to train on batches as in tensorflow, use model.train_on_batch(), model.test_on_batch(), model.predict_on_batch()

Using generators or keras.utils.Sequence

- Generators, or keras.utils.Sequence are handled by keras very efficiently by means of buffers of batches and multiple CPUs

./2_keras_intro/examples/e2_a_fit_from_generator.py

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
#Rescale data.
x_train= x_train /255.0
x_test=  x_test  /255.0
print(x_train.shape, y_train.shape, x_train.max(), x_train.min() )
#One-hot-encode labels
oh_encoder= OneHotEncoder(n_values=10, sparse=False)
y_train= oh_encoder.fit_transform(y_train)
y_test= oh_encoder.transform(y_test)

x_train, x_validation, y_train, y_validation = cross_validation.train_test_split(
                                                x_train, y_train, test_size=0.1, random_state=121)

DATA_MANAGER_OPTION=1 #Can be 1 or 2

#OPTION 1
def generateBatches(data, labels):
  '''
  This is a dummy example, here you will be loading files,
  applying transformations, etc.
  '''
  while True:
    data, labels= shuffle(data, labels)
    for i in range(0, data.shape[0], BATCH_SIZE):
      batch_x= data[i:i+BATCH_SIZE, ...]
      #transform batch_x or apply data augmentation here
      batch_y= labels[i:i+BATCH_SIZE, ...]
      yield batch_x, batch_y
      #The following line is a possible real example in which data are filenames
      #return np.array([ resize(imread(file_name), (200, 200))
      #          for file_name in batch_x]), np.array(batch_y)

trainGenerator= generateBatches(x_train, y_train)
nBatchesPerEpochTrain= x_train.shape[0]//BATCH_SIZE
validationGenerator= generateBatches(x_validation, y_validation)
nBatchesPerEpochValid= x_validation.shape[0]//BATCH_SIZE
testGenerator= generateBatches(x_test, y_test)
nBatchesPerEpochTest= x_test.shape[0]//BATCH_SIZE
```

```
#OPTION 2
class myDataManger(Sequence):
  def __init__(self, x_set, y_set, batch_size= BATCH_SIZE):
    self.x, self.y = x_set, y_set
    self.batch_size = batch_size

  def __len__(self):
    return int(np.ceil(len(self.x) / float(self.batch_size)))

  def __getitem__(self, idx):
    '''
      idx is the index of a given example
    '''
    batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
    batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]
    return batch_x, batch_y
```

```
#Model training
if DATA_MANAGER_OPTION==1:
  #OPTION 1
  model.fit_generator(trainGenerator, steps_per_epoch=nBatchesPerEpochTrain, epochs=N_EPOCHS,
                      validation_data= validationGenerator, validation_steps=nBatchesPerEpochValid,
                      workers=2, max_queue_size=10, use_multiprocessing=True )
if DATA_MANAGER_OPTION==2:
  #OPTION 2
  model.fit_generator(myDataManger(x_train, y_train), epochs=N_EPOCHS,
                      validation_data= myDataManger(x_validation, y_validation),
                      workers=2, max_queue_size=10 )
```

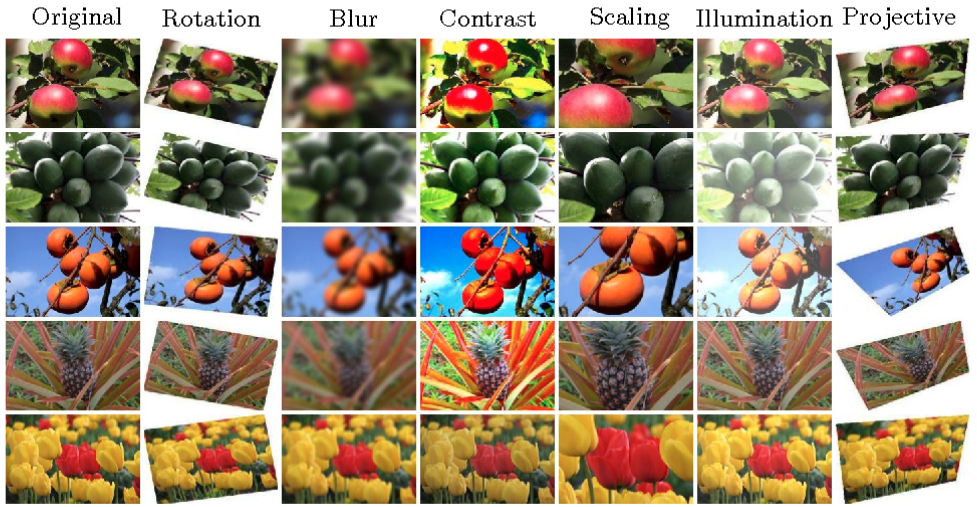
Data augmentation

./2_keras_intro/examples/e2_b_dataAugmentation.py

For images or sequences you can use built-in classes.

https://keras.io/preprocessing/image/#imagedatagenerator-methods

```
30
31 if AUGMENTATION_OPTION==1:
32   #OPTION 1: AUTOMATIC AUGMENTATION. Data need to fit in memory or being stored in one directory per class
33   #Image augmentation
34   dataAugmentator = ImageDataGenerator(rotation_range=10.0, horizontal_flip=True, fill_mode='reflect')
112
113 if AUGMENTATION_OPTION==1:
114   #When data is located as png, jpg, bmp, files ImageDataGenerator can use then directly with
115   #dataAugmentator.flow_from_directory
116   model.fit_generator(dataAugmentator.flow(x_train, y_train), epochs=N_EPOCHS,
117                       validation_data= (x_validation, y_validation),
118                       workers=2, max_queue_size=10, use_multiprocessing=True )
```



Data augmentation

For other types of data you have to handle it on your way.

- Easy when coupled with generators or keras.utils.Sequence.

./2_keras_intro/examples/e2_b_dataAugmentation.py

```python
#OPTION 2: MANUAL AUGMENTATION. If you want to use generators
from skimage.transform import rotate
class myDataManger(Sequence):
  def __init__(self, x_set, y_set, batch_size= BATCH_SIZE, augment=True):
    self.x, self.y = x_set, y_set
    self.batch_size = batch_size
    if augment:
      self.augmentBatch= self._augmentBatch
    else:
      self.augmentBatch= lambda arg: arg

  def __len__(self):
    return int(np.ceil(len(self.x) / float(self.batch_size)))

  def _random_flip_leftright(self, batch):
    for i in range(len(batch)):
      if bool(random.getrandbits(1)):
        batch[i] = np.fliplr(batch[i])
    return batch

  def _random_flip_updown(self, batch):
    for i in range(len(batch)):
      if bool(random.getrandbits(1)):
        batch[i] = np.flipud(batch[i])
    return batch

  def _random_rotation(self, batch, max_angle=15.0):
    for i in range(len(batch)):
      if bool(random.getrandbits(1)):
        # Random angle
        angle = random.uniform(-max_angle, max_angle)
        batch[i] = rotate(batch[i], angle, mode="reflect")
    return batch

  def _augmentBatch(self, batch):
    if bool(random.getrandbits(1)):
      batch= self._random_flip_leftright(batch)
    if bool(random.getrandbits(1)):
      batch= self._random_rotation(batch, 10.0)
    return batch

  def __getitem__(self, idx):
    '''
      idx is the index of a given example
    '''
    batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
    batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]
    return self.augmentBatch(batch_x), batch_y

```

Model definition

Two alternatives:

- Easy models (Feed-forward, no bifurcations, etc.)
  - Use Sequential API (https://keras.io/models/sequential/)
- Complicate models
  - Use Functional API (https://keras.io/models/model/)

Sequential API

- keras.models.Sequential class

```
42 model = keras.models.Sequential()
43 model.add( layers.Conv2D(filters=32, kernel_size= 5, strides=1,padding='same',
44                          activation='relu',  input_shape= x_train.shape[1:] ))
45 model.add( layers.MaxPooling2D(pool_size=4, strides=2 ) )
46 model.add( layers.Flatten() )
47 model.add( layers.Dense(N_HIDDEN, activation='relu'))
48 model.add( layers.Dense(y_train.shape[-1], activation='softmax'))
49 sgd= keras.optimizers.SGD(lr=LEARNING_RATE, decay=0., momentum=0., nesterov=False)
50
51 model.compile(loss='categorical_crossentropy',optimizer=sgd, metrics=['accuracy', roc_auc])
52 print("network created")
```

https://keras.io/

./2_keras_intro/examples/e1_Sequential_keras.py

Functional API

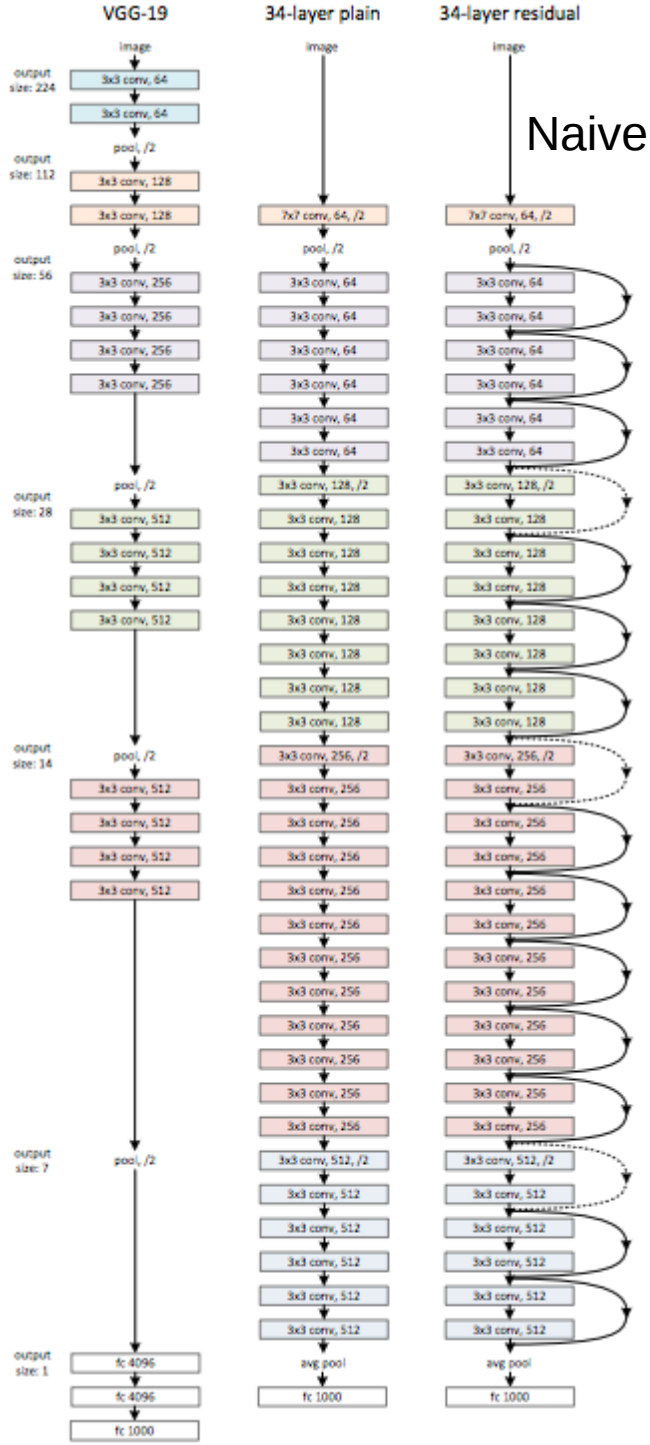
- keras.models.Model class

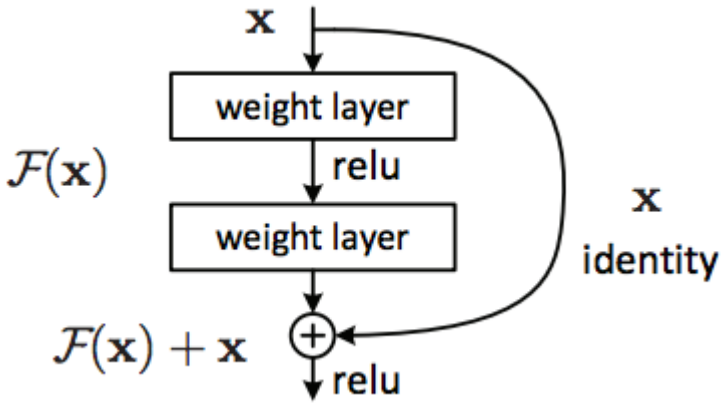
```
40 #Model definition
41
42 net_input= layers.Input( shape= x_train.shape[1:])
43 net_out= layers.Conv2D(filters=32, kernel_size= 5, strides=1,padding='same',
44                        activation='relu'  )(net_input)
45 net_out= layers.MaxPooling2D(pool_size=4, strides=2 )(net_out)
46 net_out= layers.Flatten()(net_out)
47 net_out= layers.Dense(N_HIDDEN, activation='relu')(net_out)
48 net_out= layers.Dense(y_train.shape[-1], activation='softmax')(net_out)
49
50 model = keras.models.Model( inputs=net_input, outputs= net_out )
51
52 sgd= keras.optimizers.SGD(lr=LEARNING_RATE, decay=0., momentum=0., nesterov=False)
53
54 model.compile(loss='categorical_crossentropy',optimizer=sgd, metrics=['accuracy', roc_auc])
55 print("network created")
```

https://keras.io/

./2_keras_intro/examples/e3_Funtional_keras.py

Bifurcation example: Residual Network

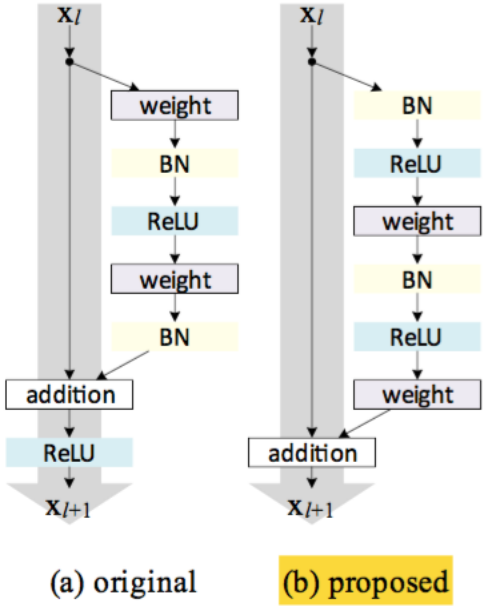




https://arxiv.org/pdf/1512.03385.pdf

./2_keras_intro/examples/e4_a_Residual_keras.py

Bifurcation example: Residual Network



https://arxiv.org/pdf/1512.03385.pdf

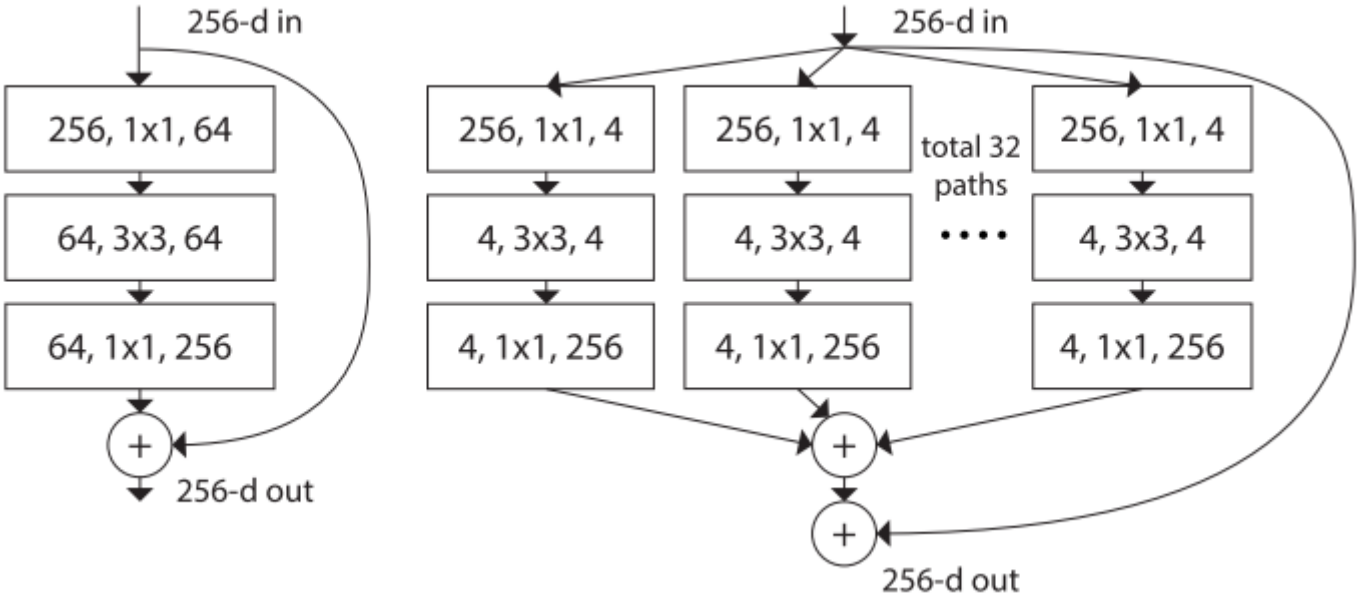
We will be suffering overfitting:

- Dropout
- Regularization (l1 or l2)
- Data augmentation

Batch Normalization (BN) makes training faster, and it may slightly reduce overfitting.

./2_keras_intro/examples/e4_b_Residual_keras_better.py

Your turn: ResNext like block

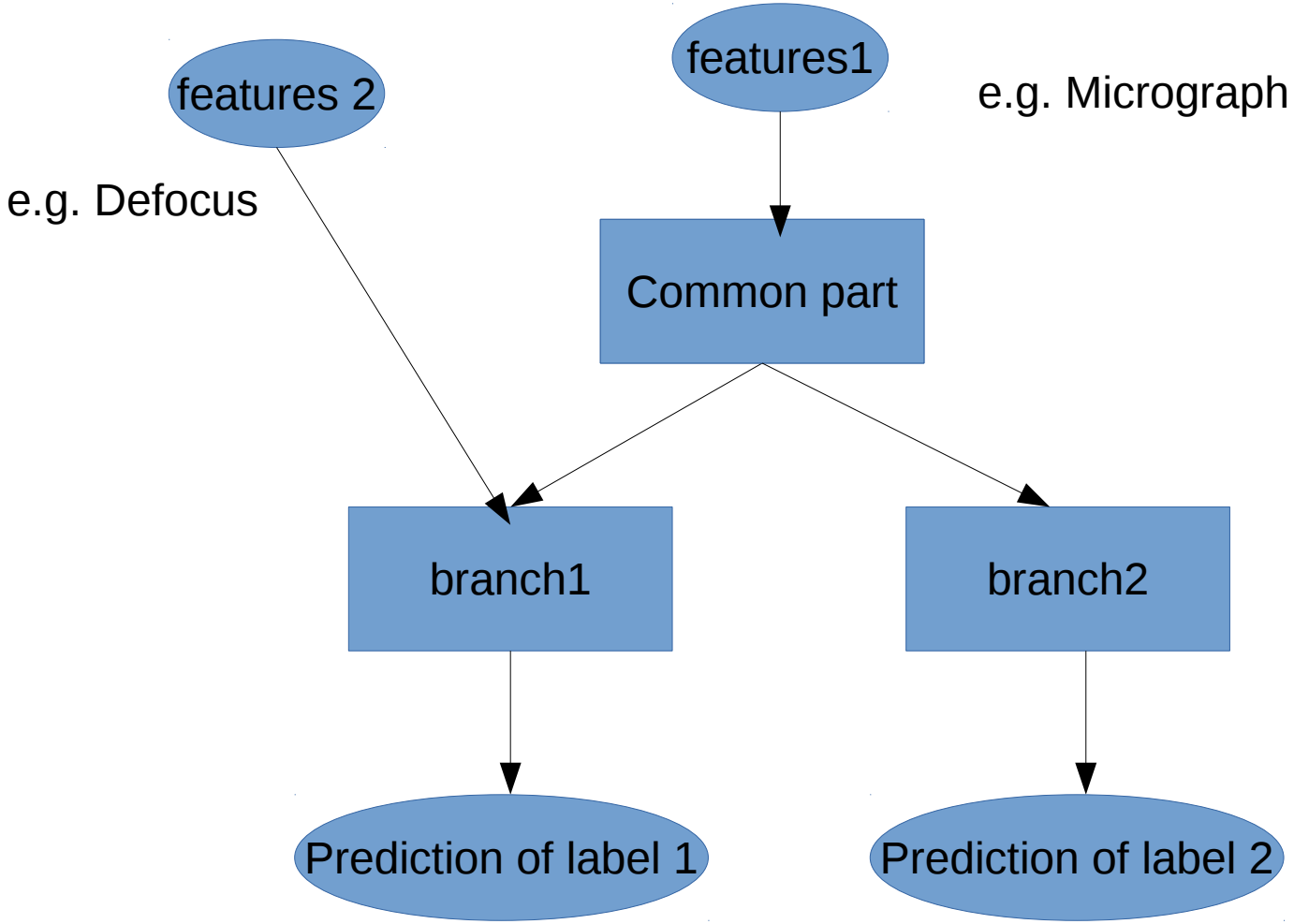


ResNext block

https://arxiv.org/pdf/1611.05431.pdf

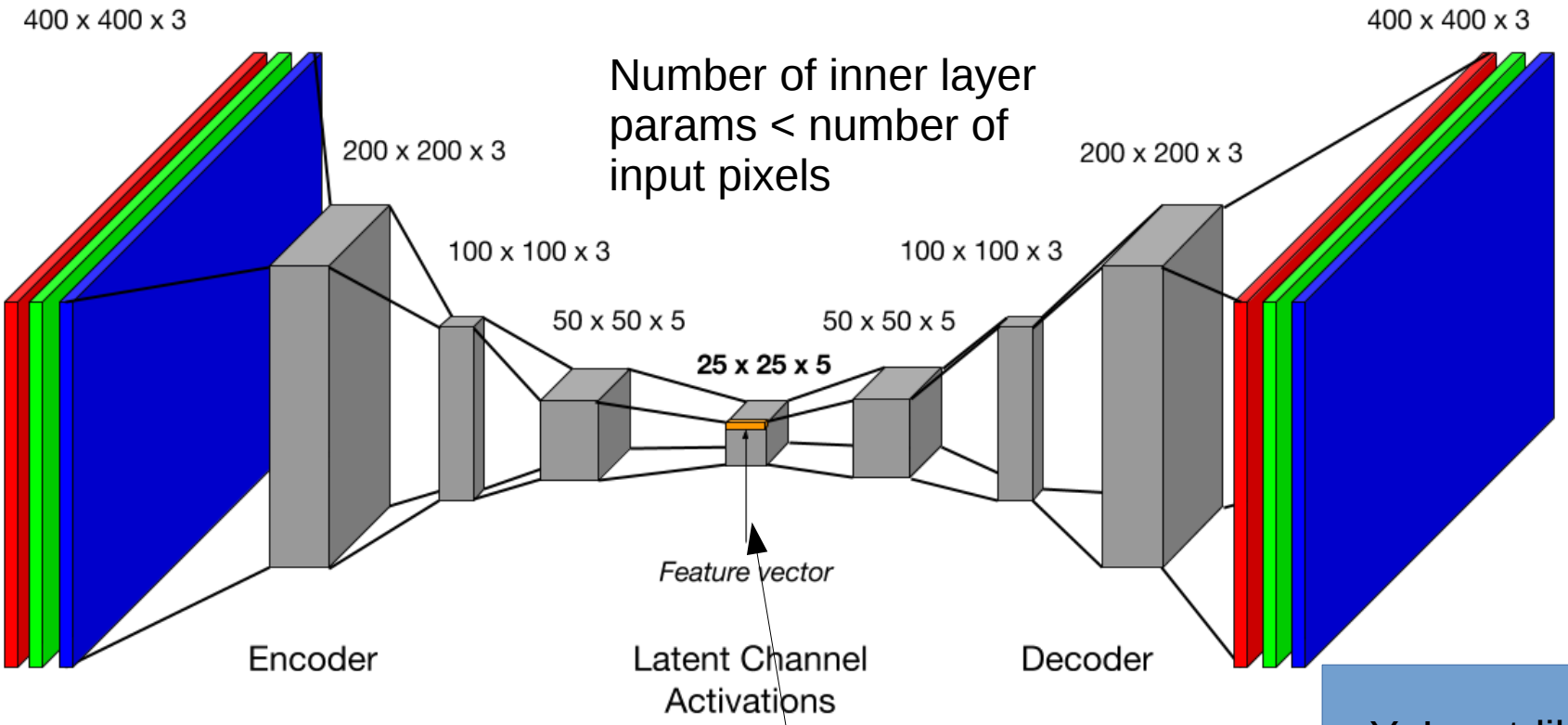
./2_keras_intro/exercises_sol/sol1_ResNext.py

Several inputs and/or outputs models



./2_keras_intro/examples/e5_severalInputsOutputs.py

Your turn: implement autoencoder



Number of inner layer params < number of input pixels

X: Input

Y: Input-like output

Z: Compressed output

Cost= mean$(X-Y)^2$

Denoising autoencoders add gaussian noise to X in order to force better representation

./2_keras_intro/exercises_sol/sol2_convAutoencoder.py

Using low level keras: keras.backend

- Keras.backend is similar to tf low level calls.
- Used inside Lambda layers and loss or metrics definitions.
  - Direct tf calls can be done inside

```
34 def roc_auc(y_true, y_pred):
35   auc = tf.metrics.auc(y_true, y_pred)[1]
36   keras.backend.get_session().run(tf.local_variables_initializer())
37   return auc
```

```
# add a x -> x^2 layer
model.add(Lambda(lambda x: x ** 2))
```

```
# add a layer that returns the concatenation
# of the positive part of the input and
# the opposite of the negative part

def antirectifier(x):
    x -= K.mean(x, axis=1, keepdims=True)
    x = K.l2_normalize(x, axis=1)
    pos = K.relu(x)
    neg = K.relu(-x)
    return K.concatenate([pos, neg], axis=1)

def antirectifier_output_shape(input_shape):
    shape = list(input_shape)
    assert len(shape) == 2  # only valid for 2D tensors
    shape[-1] *= 2
    return tuple(shape)

model.add(Lambda(antirectifier,
                 output_shape=antirectifier_output_shape))
```

Training tricks: callbacks

Callbacks are functions that are applied at certain stages of training, typically at the end or the beginning of a new epoch. Passed as argument to model.fit()

Very useful default callbacks:

- keras.callbacks.ModelCheckpoint.
- keras.callbacks.EarlyStopping.
- keras.callbacks.ReduceLROnPlateau
- keras.callbacks.TensorBoard

./2_keras_intro/examples/e6_model_with_callbacks.py

keras.callbacks.ModelCheckpoint saves keras model after epochs that have improved evaluation metric.

./2_keras_intro/examples/e6_model_with_callbacks.py

Save model manually

```
52
53 from keras.models import load_model
54 model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
55 del model  # deletes the existing model
56 # returns a compiled model
57 # identical to the previous one
58 model = load_model('my_model.h5')
59
```

Checkpointer saving model

```
51
52
53 from keras.callbacks import ModelCheckpoint
54
55 checkpointer = ModelCheckpoint(filepath='my_model.h5', save_best_only=True)
56 model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=0,
57           validation_data=(X_val, Y_val), callbacks=[checkpointer])
58
59 del model  # deletes the existing model
60 model = load_model('my_model.h5')
61
```

```
keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
save_best_only=False, save_weights_only=False, mode='auto', period=1)
```

ReduceLROnPlateau helps optimizer to improve results by lowering learning rate when stuck

```
keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=10, verbose=0, mode='auto', min_delta=0.0001, cooldown=0,
min_lr=0)
```

EarlyStopping prevents overfitting by stopping when error is not improving

```
keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0,
verbose=0, mode='auto', baseline=None)
```

Tensorboard usage

```
keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0, batch_size=32,
write_graph=True, write_grads=False, write_images=False, embeddings_freq=0,
embeddings_layer_names=None, embeddings_metadata=None, embeddings_data=None)
```

```
101
102 tf.summary.image("input images", model.get_layer("initial_conv").input)
103 tf.summary.image("first layer output filter0", model.get_layer("initial_conv").output[...,0:1]) #input must be a 4-D tensor with last dimension 1 or 3
104
105 callbacksList+= [keras.callbacks.TensorBoard(log_dir='./logs', batch_size=BATCH_SIZE, histogram_freq=1, write_graph=True, write_grads=True, write_images=False)]
```

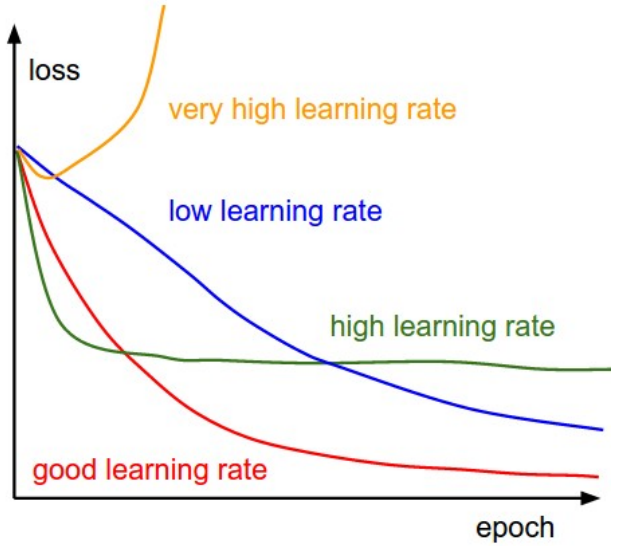
Warning: Makes training slower specially if write_xx=True. Use only for debugging.

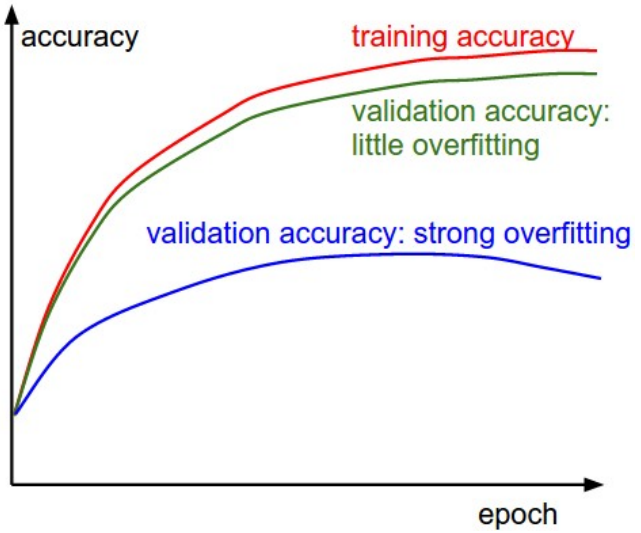
If True all layers will be recorded, better with tf.summary

Your turn: do your best on cifar10

My advices:

- Always normalize data
- First try to overfit small subset of training data using big network. If you are not able to do so, there is some problem with your data or your network.
- Use Adam optimizer. If loss oscillates, try smaller learning rate. If keeps vibrating, try different values of epsilon parameter. Increasing batch size also may help.
- Use ReduceLROnPlateau.
- Use checkpoints and employ the one with the best performance. (Sometimes using several checkpoints and averaging them works better).
- Start with a well established architecture.
- Monitor train and validation loss or accuracy. Diagnose weather you are suffering overfitting or underfitting and apply corrective actions (more/less regularization, data augmentation…).





The end