cats_and_dogs

March 4, 2023

[1]:
```
import tensorflow as tf
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('dark_background')
import numpy as np
import tensorflow_datasets as tfds
import tensorflow_addons as tfa
import os
import random
```

[2]:
```
(train_ds, validation_ds, test_ds), metadata = tfds.load('cats_vs_dogs',
                                                         split=['train[:80%]',␣
 ↪'train[80%:90%]','train[90%:]'],
                                                         with_info=True,
                                                         as_supervised=True)
```

```
Metal device set to: Apple M1 Pro

systemMemory: 16.00 GB
maxCacheSize: 5.33 GB


2023-03-04 15:26:57.371182: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2023-03-04 15:26:57.371320: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

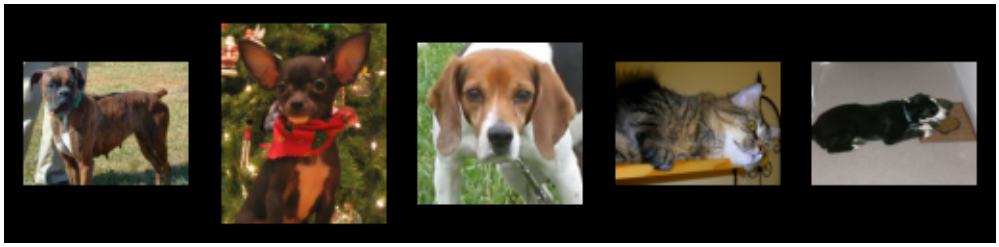
[3]:
```
dataset = train_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64),␣
 ↪label)
    ).take(5)
```

[4]:
```
f, axarr = plt.subplots(1,5)

i = 0
for image, label in dataset:
  print('Image shape: ', np.shape(image))
  tf.print('Label: ', label)
  axarr[i].imshow(image)
  axarr[i].axis('off')
  i += 1
```

```
Image shape:  (262, 350, 3)
Label:  1
Image shape:  (409, 336, 3)
Label:  1
Image shape:  (493, 500, 3)
Label:  1
Image shape:  (375, 500, 3)
Label:  0
Image shape:  (240, 320, 3)
Label:  1
```

```
2023-03-04 15:27:03.271220: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
```



[5]:
```
rnd = np.random.RandomState(seed=1234)
# we need to convert images to same shape
# we need to convert images to same shape
IMG_HEIGHT = 224
IMG_WIDTH = 224

train = train_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64),␣
 ↪label)
).map( # resize for input
      lambda image, label: (tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH]),␣
 ↪label)
).map( # random horizontal fip
```

```
      lambda image, label: (tf.image.random_flip_left_right(image, seed=rnd.
 ↪seed(1234)), label)
).map( # random translation
      lambda image, label: (tfa.image.translate(image, tf.random.
 ↪normal(shape=[2], seed=1234)), label)
).take(5)
```

[6]:
```
f, axarr = plt.subplots(1,5)

i = 0
for image, label in train:
  print('Image shape: ', np.shape(image))
  tf.print('Label:',label)
  axarr[i].imshow(image)
  axarr[i].axis('off')
  i += 1
```

```
Image shape:  (224, 224, 3)
Label: 1
Image shape:  (224, 224, 3)
Label: 1
Image shape:  (224, 224, 3)
Label: 1
Image shape:  (224, 224, 3)
Label: 0
Image shape:  (224, 224, 3)
Label: 1
```

[75]:
```
rnd = np.random.RandomState(seed=None)
num_threads = 4

# putting all the data processing pipeline on the CPU
# to make sure that the GPU is only used for training the deep neural network␣
 ↪model
with tf.device('/cpu:0'):
    train = train_ds.map(
         lambda image, label: (tf.image.convert_image_dtype(image, tf.float64),␣
 ↪label)
```

```
        ).map( # resize for input
        lambda image, label: (tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH]),␣
 ↪label)
        ).map( # random horizontal fip
        lambda image, label: (tf.image.random_flip_left_right(image, seed=rnd.
 ↪seed(1234)), label), num_parallel_calls=num_threads
        ).map( # random translation
        lambda image, label: (tfa.image.translate(image, tf.random.
 ↪normal(shape=[2], seed=rnd.seed(1234))), label),␣
 ↪num_parallel_calls=num_threads
        ).shuffle(1000).batch(32).take(1).prefetch(1)

with tf.device('/cpu:0'):
    valid = validation_ds.map(
        lambda image, label: (tf.image.convert_image_dtype(image, tf.
 ↪float64), label)
        ).map( # resize for input
        lambda image, label: (tf.image.resize(image, [IMG_HEIGHT,␣
 ↪IMG_WIDTH]), label)
        ).map( # random horizontal fip
        lambda image, label: (tf.image.random_flip_left_right(image, seed=rnd.
 ↪seed(1234)), label), num_parallel_calls=num_threads
        ).map( # random translation
        lambda image, label: (tfa.image.translate(image, tf.random.
 ↪normal(shape=[2], seed=rnd.seed(1234))), label),␣
 ↪num_parallel_calls=num_threads
        ).repeat().batch(128).take(1).cache( # cache images
        ).prefetch(1)

test = test_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64),␣
 ↪label)
).map( # resize for input
     lambda image, label: (tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH]),␣
 ↪label)
).cache( # cache images
).map( # random horizontal fip
     lambda image, label: (tfa.image.translate(image, tf.random.
 ↪normal(shape=[2], seed=rnd.seed(1234))), label)
).map( # random translation
     lambda image, label: (tf.image.rgb_to_grayscale(image), label)
).shuffle(100).prefetch(1)
```

[76]:
```
# Using pre trained MobileNetV2 model
# this model is trained on 224x224 images
mobnet = tf.keras.applications.MobileNetV2(include_top=False, weights=None)
```

```
# trainable parameters in model
nvar = len(mobnet.trainable_variables)
print('Number of variables: ', nvar)
for i in np.arange(nvar):
  print(mobnet.trainable_variables[i].name) # variable name
  print(mobnet.trainable_variables[i].shape) # kernel shape
```

```
Number of variables:  156
Conv1/kernel:0
(3, 3, 3, 32)
bn_Conv1/gamma:0
(32,)
bn_Conv1/beta:0
(32,)
expanded_conv_depthwise/depthwise_kernel:0
(3, 3, 32, 1)
expanded_conv_depthwise_BN/gamma:0
(32,)
expanded_conv_depthwise_BN/beta:0
(32,)
expanded_conv_project/kernel:0
(1, 1, 32, 16)
expanded_conv_project_BN/gamma:0
(16,)
expanded_conv_project_BN/beta:0
(16,)
block_1_expand/kernel:0
(1, 1, 16, 96)
block_1_expand_BN/gamma:0
(96,)
block_1_expand_BN/beta:0
(96,)
block_1_depthwise/depthwise_kernel:0
(3, 3, 96, 1)
block_1_depthwise_BN/gamma:0
(96,)
block_1_depthwise_BN/beta:0
(96,)
block_1_project/kernel:0
(1, 1, 96, 24)
block_1_project_BN/gamma:0
(24,)
block_1_project_BN/beta:0
(24,)
block_2_expand/kernel:0
(1, 1, 24, 144)
```

```
block_2_expand_BN/gamma:0
(144,)
block_2_expand_BN/beta:0
(144,)
block_2_depthwise/depthwise_kernel:0
(3, 3, 144, 1)
block_2_depthwise_BN/gamma:0
(144,)
block_2_depthwise_BN/beta:0
(144,)
block_2_project/kernel:0
(1, 1, 144, 24)
block_2_project_BN/gamma:0
(24,)
block_2_project_BN/beta:0
(24,)
block_3_expand/kernel:0
(1, 1, 24, 144)
block_3_expand_BN/gamma:0
(144,)
block_3_expand_BN/beta:0
(144,)
block_3_depthwise/depthwise_kernel:0
(3, 3, 144, 1)
block_3_depthwise_BN/gamma:0
(144,)
block_3_depthwise_BN/beta:0
(144,)
block_3_project/kernel:0
(1, 1, 144, 32)
block_3_project_BN/gamma:0
(32,)
block_3_project_BN/beta:0
(32,)
block_4_expand/kernel:0
(1, 1, 32, 192)
block_4_expand_BN/gamma:0
(192,)
block_4_expand_BN/beta:0
(192,)
block_4_depthwise/depthwise_kernel:0
(3, 3, 192, 1)
block_4_depthwise_BN/gamma:0
(192,)
block_4_depthwise_BN/beta:0
(192,)
block_4_project/kernel:0
(1, 1, 192, 32)
```

```
block_4_project_BN/gamma:0
(32,)
block_4_project_BN/beta:0
(32,)
block_5_expand/kernel:0
(1, 1, 32, 192)
block_5_expand_BN/gamma:0
(192,)
block_5_expand_BN/beta:0
(192,)
block_5_depthwise/depthwise_kernel:0
(3, 3, 192, 1)
block_5_depthwise_BN/gamma:0
(192,)
block_5_depthwise_BN/beta:0
(192,)
block_5_project/kernel:0
(1, 1, 192, 32)
block_5_project_BN/gamma:0
(32,)
block_5_project_BN/beta:0
(32,)
block_6_expand/kernel:0
(1, 1, 32, 192)
block_6_expand_BN/gamma:0
(192,)
block_6_expand_BN/beta:0
(192,)
block_6_depthwise/depthwise_kernel:0
(3, 3, 192, 1)
block_6_depthwise_BN/gamma:0
(192,)
block_6_depthwise_BN/beta:0
(192,)
block_6_project/kernel:0
(1, 1, 192, 64)
block_6_project_BN/gamma:0
(64,)
block_6_project_BN/beta:0
(64,)
block_7_expand/kernel:0
(1, 1, 64, 384)
block_7_expand_BN/gamma:0
(384,)
block_7_expand_BN/beta:0
(384,)
block_7_depthwise/depthwise_kernel:0
(3, 3, 384, 1)
```

```
block_7_depthwise_BN/gamma:0
(384,)
block_7_depthwise_BN/beta:0
(384,)
block_7_project/kernel:0
(1, 1, 384, 64)
block_7_project_BN/gamma:0
(64,)
block_7_project_BN/beta:0
(64,)
block_8_expand/kernel:0
(1, 1, 64, 384)
block_8_expand_BN/gamma:0
(384,)
block_8_expand_BN/beta:0
(384,)
block_8_depthwise/depthwise_kernel:0
(3, 3, 384, 1)
block_8_depthwise_BN/gamma:0
(384,)
block_8_depthwise_BN/beta:0
(384,)
block_8_project/kernel:0
(1, 1, 384, 64)
block_8_project_BN/gamma:0
(64,)
block_8_project_BN/beta:0
(64,)
block_9_expand/kernel:0
(1, 1, 64, 384)
block_9_expand_BN/gamma:0
(384,)
block_9_expand_BN/beta:0
(384,)
block_9_depthwise/depthwise_kernel:0
(3, 3, 384, 1)
block_9_depthwise_BN/gamma:0
(384,)
block_9_depthwise_BN/beta:0
(384,)
block_9_project/kernel:0
(1, 1, 384, 64)
block_9_project_BN/gamma:0
(64,)
block_9_project_BN/beta:0
(64,)
block_10_expand/kernel:0
(1, 1, 64, 384)
```

```
block_10_expand_BN/gamma:0
(384,)
block_10_expand_BN/beta:0
(384,)
block_10_depthwise/depthwise_kernel:0
(3, 3, 384, 1)
block_10_depthwise_BN/gamma:0
(384,)
block_10_depthwise_BN/beta:0
(384,)
block_10_project/kernel:0
(1, 1, 384, 96)
block_10_project_BN/gamma:0
(96,)
block_10_project_BN/beta:0
(96,)
block_11_expand/kernel:0
(1, 1, 96, 576)
block_11_expand_BN/gamma:0
(576,)
block_11_expand_BN/beta:0
(576,)
block_11_depthwise/depthwise_kernel:0
(3, 3, 576, 1)
block_11_depthwise_BN/gamma:0
(576,)
block_11_depthwise_BN/beta:0
(576,)
block_11_project/kernel:0
(1, 1, 576, 96)
block_11_project_BN/gamma:0
(96,)
block_11_project_BN/beta:0
(96,)
block_12_expand/kernel:0
(1, 1, 96, 576)
block_12_expand_BN/gamma:0
(576,)
block_12_expand_BN/beta:0
(576,)
block_12_depthwise/depthwise_kernel:0
(3, 3, 576, 1)
block_12_depthwise_BN/gamma:0
(576,)
block_12_depthwise_BN/beta:0
(576,)
block_12_project/kernel:0
(1, 1, 576, 96)
```

```
block_12_project_BN/gamma:0
(96,)
block_12_project_BN/beta:0
(96,)
block_13_expand/kernel:0
(1, 1, 96, 576)
block_13_expand_BN/gamma:0
(576,)
block_13_expand_BN/beta:0
(576,)
block_13_depthwise/depthwise_kernel:0
(3, 3, 576, 1)
block_13_depthwise_BN/gamma:0
(576,)
block_13_depthwise_BN/beta:0
(576,)
block_13_project/kernel:0
(1, 1, 576, 160)
block_13_project_BN/gamma:0
(160,)
block_13_project_BN/beta:0
(160,)
block_14_expand/kernel:0
(1, 1, 160, 960)
block_14_expand_BN/gamma:0
(960,)
block_14_expand_BN/beta:0
(960,)
block_14_depthwise/depthwise_kernel:0
(3, 3, 960, 1)
block_14_depthwise_BN/gamma:0
(960,)
block_14_depthwise_BN/beta:0
(960,)
block_14_project/kernel:0
(1, 1, 960, 160)
block_14_project_BN/gamma:0
(160,)
block_14_project_BN/beta:0
(160,)
block_15_expand/kernel:0
(1, 1, 160, 960)
block_15_expand_BN/gamma:0
(960,)
block_15_expand_BN/beta:0
(960,)
block_15_depthwise/depthwise_kernel:0
(3, 3, 960, 1)
```

```
block_15_depthwise_BN/gamma:0
(960,)
block_15_depthwise_BN/beta:0
(960,)
block_15_project/kernel:0
(1, 1, 960, 160)
block_15_project_BN/gamma:0
(160,)
block_15_project_BN/beta:0
(160,)
block_16_expand/kernel:0
(1, 1, 160, 960)
block_16_expand_BN/gamma:0
(960,)
block_16_expand_BN/beta:0
(960,)
block_16_depthwise/depthwise_kernel:0
(3, 3, 960, 1)
block_16_depthwise_BN/gamma:0
(960,)
block_16_depthwise_BN/beta:0
(960,)
block_16_project/kernel:0
(1, 1, 960, 320)
block_16_project_BN/gamma:0
(320,)
block_16_project_BN/beta:0
(320,)
Conv_1/kernel:0
(1, 1, 320, 1280)
Conv_1_bn/gamma:0
(1280,)
Conv_1_bn/beta:0
(1280,)
```

[77]:
```
# clear keras session
tf.keras.backend.clear_session()
```

0.1 Training from scratch

We are going to add a dropout layer and a new dense layer for 2 classes (i.e., cats and dogs)

[78]:
```
dropout_rate = 0.2
num_classes = 2
CHANS = 3 # number of channels in input images
input_shape = [IMG_HEIGHT, IMG_WIDTH, CHANS]
```

```
mbnet = tf.keras.applications.MobileNetV2(input_shape=input_shape,␣
 ↪include_top=False, weights=None)

model = tf.keras.Sequential([
  mbnet,
  tf.keras.layers.GlobalAveragePooling2D(),
  tf.keras.layers.Dropout(dropout_rate),
  tf.keras.layers.Dense(num_classes, activation='softmax')
], name='pre-trained_mobilenetV2')
model.build()

model.summary()
```

```
Model: "pre-trained_mobilenetV2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,226,434
Non-trainable params: 34,112
_________________________________________________________________
```

[79]:
```
with tf.GradientTape() as tape:
  tape.reset()
```

[81]:
```
LR = 0.001 # learning rate
optimizer = tf.optimizers.Adam(learning_rate=LR) # adam optimizer

def train_step(model, X, Y):
  with tf.GradientTape() as tape:
    pred = model(X)
    current_loss = tf.reduce_mean(tf.losses.categorical_crossentropy(Y, pred))
  grads = tape.gradient(current_loss, model.trainable_variables)
  optimizer.apply_gradients(zip( grads , model.trainable_variables)) # update␣
 ↪gradients of all layers
  current_accuracy = tf.reduce_mean(tf.metrics.categorical_accuracy(Y, pred))
```

```
  return(current_loss, current_accuracy)
```

[82]:
```
import warnings
warnings.filterwarnings('ignore')
```

[83]:
```
niter = 1000

tloss = []
tacc = []
vloss = []
vacc = []

for it in range(niter):
    for image, label in train:
        loss, acc = train_step( model , image , tf.one_hot(label, depth=2) )␣
 ↪#run training

    if it % 10 == 0: #log training metrics
      tf.print('iter: ',it, ', loss: {:.3f}, acc: {:.3f}'.format(loss, acc))
      tloss.append(loss)
      tacc.append(acc)

    if it % 50 == 0: #log validation metrics
      for val_image, val_label in valid:
        val_pred = model(val_image)
        val_loss = tf.reduce_mean(tf.losses.categorical_crossentropy(tf.
 ↪one_hot(val_label, depth=2), val_pred))
        val_acc = tf.reduce_mean(tf.metrics.categorical_accuracy(tf.
 ↪one_hot(val_label, depth=2), val_pred))
        tf.print('iter: ',it, ', validation loss: {:.3f}, validation acc: {:.
 ↪3f}'.format(val_loss, val_acc))
        vloss.append(val_loss)
        vacc.append(val_acc)
```

```
iter:  0 , loss: 0.693, acc: 0.438
iter:  0 , validation loss: 0.691, validation acc: 0.555
iter:  10 , loss: 0.693, acc: 0.531
iter:  20 , loss: 0.693, acc: 0.531
iter:  30 , loss: 0.693, acc: 0.500
iter:  40 , loss: 0.694, acc: 0.469
iter:  50 , loss: 0.691, acc: 0.594
iter:  50 , validation loss: 0.695, validation acc: 0.445
iter:  60 , loss: 0.693, acc: 0.500
iter:  70 , loss: 0.686, acc: 0.594
iter:  80 , loss: 0.689, acc: 0.562
iter:  90 , loss: 0.690, acc: 0.562
iter:  100 , loss: 0.690, acc: 0.562
```

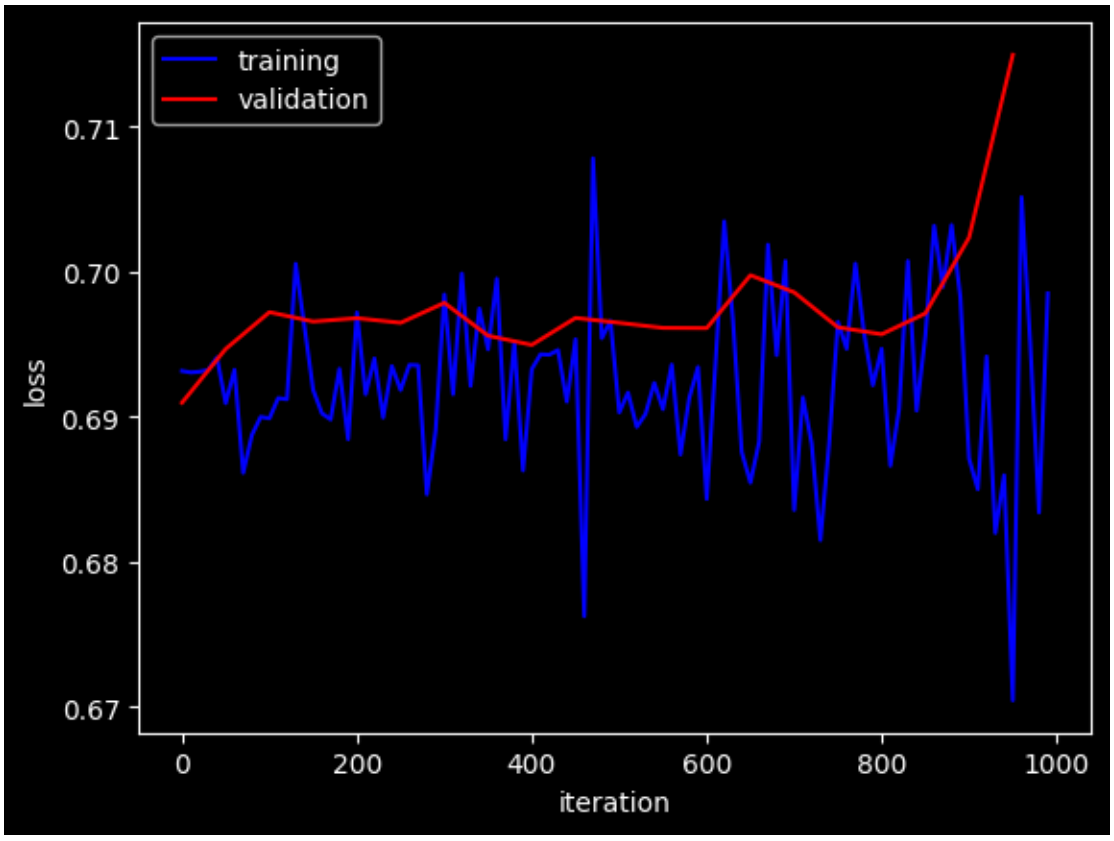
```
iter:  100 , validation loss: 0.697, validation acc: 0.445
iter:  110 , loss: 0.691, acc: 0.531
iter:  120 , loss: 0.691, acc: 0.531
iter:  130 , loss: 0.701, acc: 0.406
iter:  140 , loss: 0.696, acc: 0.438
iter:  150 , loss: 0.692, acc: 0.531
iter:  150 , validation loss: 0.697, validation acc: 0.445
iter:  160 , loss: 0.690, acc: 0.562
iter:  170 , loss: 0.690, acc: 0.594
iter:  180 , loss: 0.693, acc: 0.500
iter:  190 , loss: 0.688, acc: 0.625
iter:  200 , loss: 0.697, acc: 0.438
iter:  200 , validation loss: 0.697, validation acc: 0.445
iter:  210 , loss: 0.692, acc: 0.531
iter:  220 , loss: 0.694, acc: 0.500
iter:  230 , loss: 0.690, acc: 0.562
iter:  240 , loss: 0.693, acc: 0.500
iter:  250 , loss: 0.692, acc: 0.531
iter:  250 , validation loss: 0.696, validation acc: 0.445
iter:  260 , loss: 0.694, acc: 0.500
iter:  270 , loss: 0.694, acc: 0.500
iter:  280 , loss: 0.685, acc: 0.656
iter:  290 , loss: 0.689, acc: 0.562
iter:  300 , loss: 0.698, acc: 0.438
iter:  300 , validation loss: 0.698, validation acc: 0.445
iter:  310 , loss: 0.692, acc: 0.531
iter:  320 , loss: 0.700, acc: 0.375
iter:  330 , loss: 0.692, acc: 0.531
iter:  340 , loss: 0.697, acc: 0.406
iter:  350 , loss: 0.695, acc: 0.469
iter:  350 , validation loss: 0.696, validation acc: 0.445
iter:  360 , loss: 0.700, acc: 0.375
iter:  370 , loss: 0.688, acc: 0.594
iter:  380 , loss: 0.695, acc: 0.469
iter:  390 , loss: 0.686, acc: 0.656
iter:  400 , loss: 0.693, acc: 0.500
iter:  400 , validation loss: 0.695, validation acc: 0.445
iter:  410 , loss: 0.694, acc: 0.469
iter:  420 , loss: 0.694, acc: 0.469
iter:  430 , loss: 0.695, acc: 0.438
iter:  440 , loss: 0.691, acc: 0.562
iter:  450 , loss: 0.695, acc: 0.469
iter:  450 , validation loss: 0.697, validation acc: 0.445
iter:  460 , loss: 0.676, acc: 0.688
iter:  470 , loss: 0.708, acc: 0.344
iter:  480 , loss: 0.695, acc: 0.469
iter:  490 , loss: 0.697, acc: 0.438
iter:  500 , loss: 0.690, acc: 0.562
```

```
iter:  500 , validation loss: 0.696, validation acc: 0.445
iter:  510 , loss: 0.692, acc: 0.531
iter:  520 , loss: 0.689, acc: 0.562
iter:  530 , loss: 0.690, acc: 0.625
iter:  540 , loss: 0.692, acc: 0.562
iter:  550 , loss: 0.691, acc: 0.562
iter:  550 , validation loss: 0.696, validation acc: 0.445
iter:  560 , loss: 0.694, acc: 0.500
iter:  570 , loss: 0.687, acc: 0.625
iter:  580 , loss: 0.691, acc: 0.562
iter:  590 , loss: 0.693, acc: 0.500
iter:  600 , loss: 0.684, acc: 0.688
iter:  600 , validation loss: 0.696, validation acc: 0.445
iter:  610 , loss: 0.694, acc: 0.500
iter:  620 , loss: 0.703, acc: 0.344
iter:  630 , loss: 0.697, acc: 0.406
iter:  640 , loss: 0.688, acc: 0.594
iter:  650 , loss: 0.685, acc: 0.594
iter:  650 , validation loss: 0.700, validation acc: 0.445
iter:  660 , loss: 0.688, acc: 0.562
iter:  670 , loss: 0.702, acc: 0.438
iter:  680 , loss: 0.694, acc: 0.500
iter:  690 , loss: 0.701, acc: 0.438
iter:  700 , loss: 0.684, acc: 0.625
iter:  700 , validation loss: 0.699, validation acc: 0.445
iter:  710 , loss: 0.691, acc: 0.531
iter:  720 , loss: 0.688, acc: 0.562
iter:  730 , loss: 0.681, acc: 0.656
iter:  740 , loss: 0.688, acc: 0.594
iter:  750 , loss: 0.697, acc: 0.438
iter:  750 , validation loss: 0.696, validation acc: 0.445
iter:  760 , loss: 0.695, acc: 0.469
iter:  770 , loss: 0.701, acc: 0.312
iter:  780 , loss: 0.696, acc: 0.438
iter:  790 , loss: 0.692, acc: 0.531
iter:  800 , loss: 0.695, acc: 0.469
iter:  800 , validation loss: 0.696, validation acc: 0.445
iter:  810 , loss: 0.687, acc: 0.656
iter:  820 , loss: 0.691, acc: 0.562
iter:  830 , loss: 0.701, acc: 0.344
iter:  840 , loss: 0.690, acc: 0.562
iter:  850 , loss: 0.696, acc: 0.469
iter:  850 , validation loss: 0.697, validation acc: 0.445
iter:  860 , loss: 0.703, acc: 0.375
iter:  870 , loss: 0.699, acc: 0.438
iter:  880 , loss: 0.703, acc: 0.406
iter:  890 , loss: 0.698, acc: 0.469
iter:  900 , loss: 0.687, acc: 0.562
```

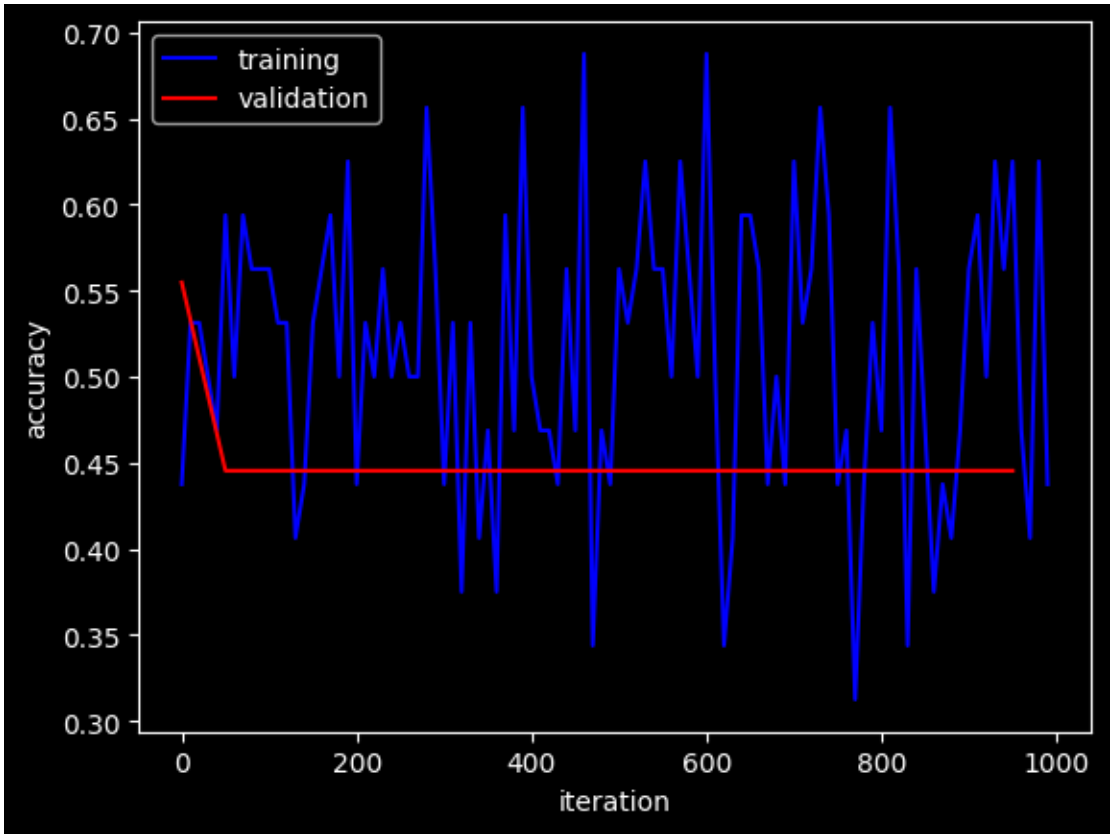
```
iter:  900 , validation loss: 0.702, validation acc: 0.445
iter:  910 , loss: 0.685, acc: 0.594
iter:  920 , loss: 0.694, acc: 0.500
iter:  930 , loss: 0.682, acc: 0.625
iter:  940 , loss: 0.686, acc: 0.562
iter:  950 , loss: 0.670, acc: 0.625
iter:  950 , validation loss: 0.715, validation acc: 0.445
iter:  960 , loss: 0.705, acc: 0.469
iter:  970 , loss: 0.695, acc: 0.406
iter:  980 , loss: 0.683, acc: 0.625
iter:  990 , loss: 0.698, acc: 0.438
```

[84]:
```
titers = 10*np.arange(np.shape(tloss)[0])
viters = 50*np.arange(np.shape(vloss)[0])

plt.plot(titers, tloss, c='b', label='training');
plt.plot(viters, vloss, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('loss');
```



[85]:
```
plt.plot(titers, tacc, c='b', label='training');
plt.plot(viters, vacc, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('accuracy');
```



0.2 Transfer learning

Now we will implement transfer learning from the imagenet weights

[86]:
```
tf.keras.backend.clear_session() # clear keras session
```

[87]:
```
dropout_rate = 0.2
num_classes = 2
IMG_HEIGHT = 224
IMG_WIDTH = 224
CHANS = 3
input_shape = [IMG_HEIGHT, IMG_WIDTH, CHANS]

mobnet = tf.keras.applications.MobileNetV2(input_shape=input_shape,␣
 ↪include_top=False, weights='imagenet')
```

```
model = tf.keras.Sequential([
    mobnet,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(num_classes, activation='softmax')
], name='MobileNetV2_weights_model')

model.build()

mobnet.trainable = False # freezes the first layers to the imagenet weights

model.summary()
```

```
Model: "MobileNetV2_weights_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

[88]:
```
with tf.GradientTape() as tape:
    tape.reset() # This resets gradient tape
```

[89]:
```
LR = 1e-2 # learning rate
optimizer = tf.keras.optimizers.Adam(LR) # initilaize adam optimizer

def train_step(model, X, Y):
    with tf.GradientTape() as tape:
        pred = model( X )
        current_loss = tf.reduce_mean(tf.losses.categorical_crossentropy( Y, ␣
 ↪pred))
    grads = tape.gradient(current_loss, model.trainable_variables)
    optimizer.apply_gradients( zip( grads , model.trainable_variables) )
```

```
    current_accuracy = tf.reduce_mean(tf.metrics.categorical_accuracy(Y, pred))
    return(current_loss, current_accuracy)
```

[90]:
```
niter = 1000

tloss = []
tacc = []
vloss = []
vacc = []

for it in range(niter):
    for image, label in train:
        loss, acc = train_step(model , image , tf.one_hot(label,depth=2) ) #run␣
 ↪training

    if it % 10 == 0: #log training metrics
      tf.print('iter: ',it, ', loss: {:.3f}, acc: {:.3f}'.format(loss, acc))
      tloss.append(loss)
      tacc.append(acc)

    if it % 50 == 0: #log validation metrics
      for val_image, val_label in valid:
        val_pred = model(val_image)
        val_loss = tf.reduce_mean(tf.losses.categorical_crossentropy(tf.
 ↪one_hot(val_label,depth=2) , val_pred))
        val_acc = tf.reduce_mean(tf.metrics.categorical_accuracy(tf.
 ↪one_hot(val_label,depth=2) , val_pred))
        tf.print('iter: ',it, ', validation loss: {:.3f}, validation acc: {:.
 ↪3f}'.format(val_loss, val_acc))
        vloss.append(val_loss)
        vacc.append(val_acc)
```

```
iter:  0 , loss: 0.839, acc: 0.438
iter:  0 , validation loss: 0.703, validation acc: 0.695
iter:  10 , loss: 0.054, acc: 0.969
iter:  20 , loss: 0.063, acc: 0.969
iter:  30 , loss: 0.086, acc: 0.938
iter:  40 , loss: 0.023, acc: 1.000
iter:  50 , loss: 0.000, acc: 1.000
iter:  50 , validation loss: 0.003, validation acc: 1.000
iter:  60 , loss: 0.002, acc: 1.000
iter:  70 , loss: 0.001, acc: 1.000
iter:  80 , loss: 0.029, acc: 1.000
iter:  90 , loss: 0.088, acc: 0.969
iter:  100 , loss: 0.006, acc: 1.000
iter:  100 , validation loss: 0.047, validation acc: 0.984
iter:  110 , loss: 0.022, acc: 1.000
```

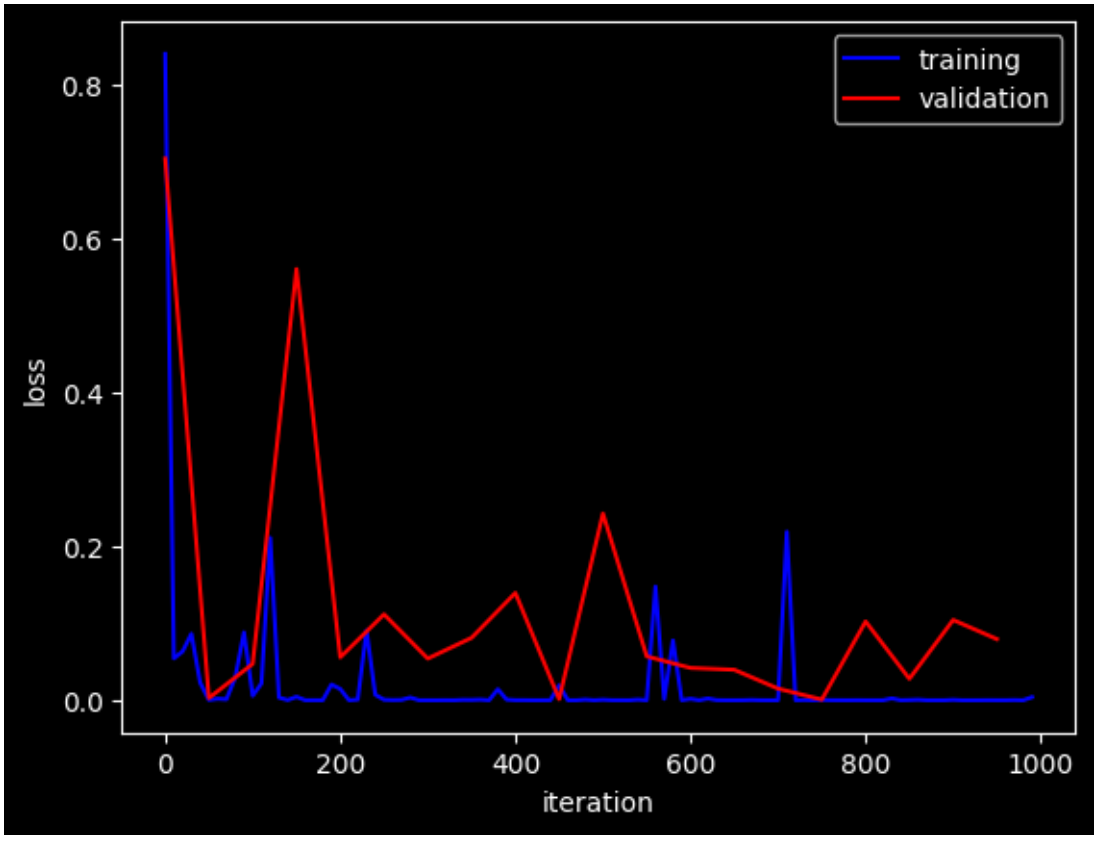
```
iter:  120 , loss: 0.211, acc: 0.938
iter:  130 , loss: 0.003, acc: 1.000
iter:  140 , loss: 0.000, acc: 1.000
iter:  150 , loss: 0.004, acc: 1.000
iter:  150 , validation loss: 0.560, validation acc: 0.914
iter:  160 , loss: 0.000, acc: 1.000
iter:  170 , loss: 0.000, acc: 1.000
iter:  180 , loss: 0.000, acc: 1.000
iter:  190 , loss: 0.020, acc: 1.000
iter:  200 , loss: 0.015, acc: 1.000
iter:  200 , validation loss: 0.056, validation acc: 0.984
iter:  210 , loss: 0.000, acc: 1.000
iter:  220 , loss: 0.001, acc: 1.000
iter:  230 , loss: 0.091, acc: 0.969
iter:  240 , loss: 0.007, acc: 1.000
iter:  250 , loss: 0.001, acc: 1.000
iter:  250 , validation loss: 0.111, validation acc: 0.984
iter:  260 , loss: 0.000, acc: 1.000
iter:  270 , loss: 0.000, acc: 1.000
iter:  280 , loss: 0.003, acc: 1.000
iter:  290 , loss: 0.000, acc: 1.000
iter:  300 , loss: 0.000, acc: 1.000
iter:  300 , validation loss: 0.054, validation acc: 0.984
iter:  310 , loss: 0.000, acc: 1.000
iter:  320 , loss: 0.000, acc: 1.000
iter:  330 , loss: 0.000, acc: 1.000
iter:  340 , loss: 0.001, acc: 1.000
iter:  350 , loss: 0.000, acc: 1.000
iter:  350 , validation loss: 0.081, validation acc: 0.984
iter:  360 , loss: 0.001, acc: 1.000
iter:  370 , loss: 0.000, acc: 1.000
iter:  380 , loss: 0.014, acc: 1.000
iter:  390 , loss: 0.001, acc: 1.000
iter:  400 , loss: 0.000, acc: 1.000
iter:  400 , validation loss: 0.139, validation acc: 0.969
iter:  410 , loss: 0.000, acc: 1.000
iter:  420 , loss: 0.000, acc: 1.000
iter:  430 , loss: 0.000, acc: 1.000
iter:  440 , loss: 0.000, acc: 1.000
iter:  450 , loss: 0.019, acc: 1.000
iter:  450 , validation loss: 0.001, validation acc: 1.000
iter:  460 , loss: 0.000, acc: 1.000
iter:  470 , loss: 0.000, acc: 1.000
iter:  480 , loss: 0.001, acc: 1.000
iter:  490 , loss: 0.000, acc: 1.000
iter:  500 , loss: 0.001, acc: 1.000
iter:  500 , validation loss: 0.242, validation acc: 0.961
iter:  510 , loss: 0.000, acc: 1.000
```

```
iter:  520 , loss: 0.000, acc: 1.000
iter:  530 , loss: 0.000, acc: 1.000
iter:  540 , loss: 0.001, acc: 1.000
iter:  550 , loss: 0.000, acc: 1.000
iter:  550 , validation loss: 0.057, validation acc: 0.992
iter:  560 , loss: 0.148, acc: 0.969
iter:  570 , loss: 0.002, acc: 1.000
iter:  580 , loss: 0.078, acc: 0.969
iter:  590 , loss: 0.000, acc: 1.000
iter:  600 , loss: 0.002, acc: 1.000
iter:  600 , validation loss: 0.042, validation acc: 0.969
iter:  610 , loss: 0.000, acc: 1.000
iter:  620 , loss: 0.002, acc: 1.000
iter:  630 , loss: 0.000, acc: 1.000
iter:  640 , loss: 0.000, acc: 1.000
iter:  650 , loss: 0.000, acc: 1.000
iter:  650 , validation loss: 0.040, validation acc: 0.984
iter:  660 , loss: 0.000, acc: 1.000
iter:  670 , loss: 0.000, acc: 1.000
iter:  680 , loss: 0.000, acc: 1.000
iter:  690 , loss: 0.000, acc: 1.000
iter:  700 , loss: 0.000, acc: 1.000
iter:  700 , validation loss: 0.015, validation acc: 0.992
iter:  710 , loss: 0.219, acc: 0.969
iter:  720 , loss: 0.000, acc: 1.000
iter:  730 , loss: 0.000, acc: 1.000
iter:  740 , loss: 0.000, acc: 1.000
iter:  750 , loss: 0.000, acc: 1.000
iter:  750 , validation loss: 0.001, validation acc: 1.000
iter:  760 , loss: 0.000, acc: 1.000
iter:  770 , loss: 0.000, acc: 1.000
iter:  780 , loss: 0.000, acc: 1.000
iter:  790 , loss: 0.000, acc: 1.000
iter:  800 , loss: 0.000, acc: 1.000
iter:  800 , validation loss: 0.102, validation acc: 0.977
iter:  810 , loss: 0.000, acc: 1.000
iter:  820 , loss: 0.000, acc: 1.000
iter:  830 , loss: 0.002, acc: 1.000
iter:  840 , loss: 0.000, acc: 1.000
iter:  850 , loss: 0.000, acc: 1.000
iter:  850 , validation loss: 0.028, validation acc: 0.992
iter:  860 , loss: 0.001, acc: 1.000
iter:  870 , loss: 0.000, acc: 1.000
iter:  880 , loss: 0.000, acc: 1.000
iter:  890 , loss: 0.000, acc: 1.000
iter:  900 , loss: 0.001, acc: 1.000
iter:  900 , validation loss: 0.104, validation acc: 0.992
iter:  910 , loss: 0.000, acc: 1.000
```

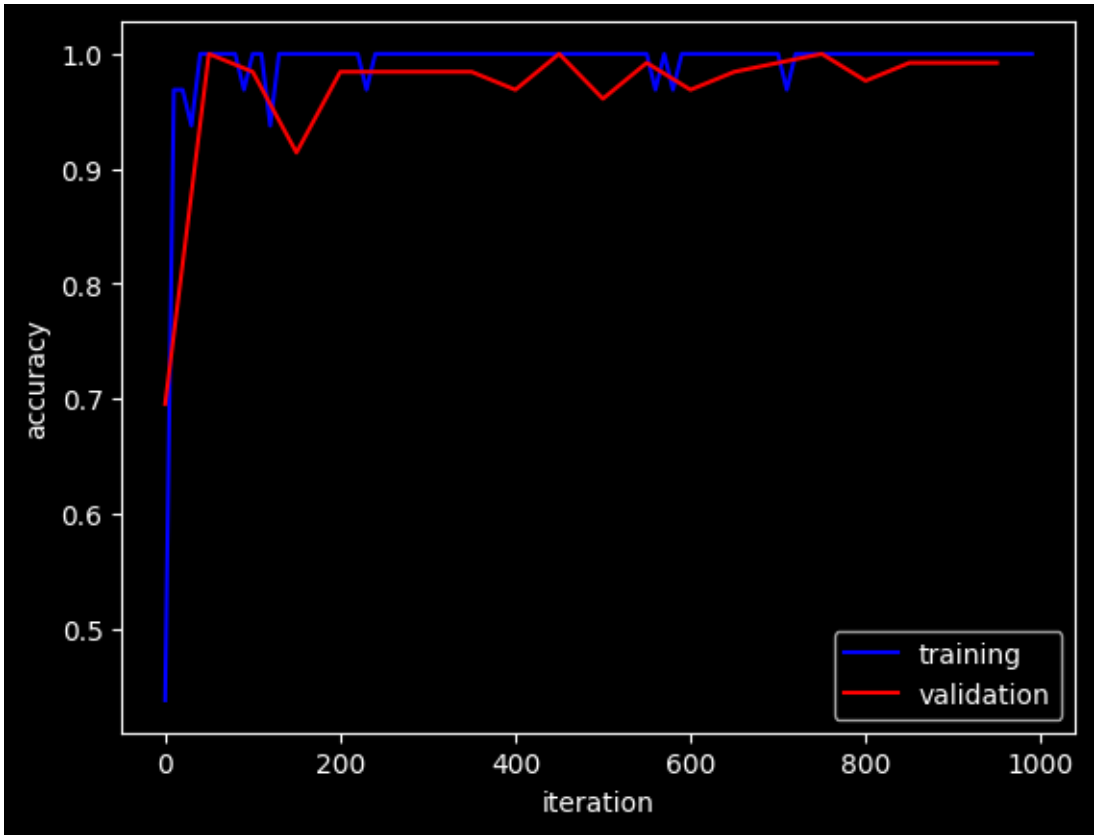
```
iter:  920 , loss: 0.000, acc: 1.000
iter:  930 , loss: 0.000, acc: 1.000
iter:  940 , loss: 0.000, acc: 1.000
iter:  950 , loss: 0.000, acc: 1.000
iter:  950 , validation loss: 0.079, validation acc: 0.992
iter:  960 , loss: 0.000, acc: 1.000
iter:  970 , loss: 0.000, acc: 1.000
iter:  980 , loss: 0.000, acc: 1.000
iter:  990 , loss: 0.004, acc: 1.000
```

[91]:
```
# plot training and validation loss as a function of iteration
titers = 10*np.arange(np.shape(tloss)[0])
viters = 50*np.arange(np.shape(vloss)[0])

plt.plot(titers, tloss, c='b', label='training');
plt.plot(viters, vloss, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('loss');
```



[92]:
```
# plot training and validation accuracy as a function of iteration
plt.plot(titers, tacc, c='b', label='training');
plt.plot(viters, vacc, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('accuracy');
```



0.3 Save weights

[93]:
```
# Lets save the trained weights into hdf5 format
import os
CODE_PATH = '/Users/rahuln/vscode/'
MODEL_PATH = os.path.join(CODE_PATH, 'mlis_workshops', 'workshop4', 'my_weights.
  ↪hdf5')
model.save_weights(MODEL_PATH)
```

[94]:
```
tf.keras.backend.clear_session()
```

0.4 Load weights

[95]:
```
# Load previously saved weights onto new model

dropoutrate = 0.2
num_classes = 2
input_shape = [IMG_HEIGHT, IMG_WIDTH, CHANS]

mobnet = tf.keras.applications.MobileNetV2(input_shape=input_shape,␣
 ↪include_top=False, weights='imagenet') #We now dont want randomized weights␣
 ↪but to load weights from imagenet

model = tf.keras.Sequential([
  mobnet,
  tf.keras.layers.GlobalAveragePooling2D(),
  tf.keras.layers.Dropout(dropoutrate),
  tf.keras.layers.Dense(num_classes, activation='softmax')
], name='loaded_sequential_model')
model.build()

mobnet.trainable = False # freeze the first layers to the imagenet weights

model.summary() # print the model
```

```
Model: "loaded_sequential_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

[96]:
```
# Fine tuning the model by training with a smaller learning rate

LR = 1e-5 # learning rate
```

```
optimizer = tf.optimizers.Adam(LR) # adam optimizer

def train_step( model, X , Y):
    with tf.GradientTape() as tape:
        pred = model( X )
        current_loss = tf.reduce_mean(tf.losses.categorical_crossentropy( Y, ␣
 ↪pred))
    grads = tape.gradient(current_loss, model.trainable_variables)
    optimizer.apply_gradients( zip( grads , model.trainable_variables) )
    current_accuracy = tf.reduce_mean(tf.metrics.categorical_accuracy(Y, pred))
    return(current_loss, current_accuracy)
```

[97]:
```
niter = 1000

tloss = []
tacc = []
vloss = []
vacc = []

for it in range(niter):
    for image, label in train:
        loss, acc = train_step( model , image , tf.one_hot(label,depth=2) )␣
 ↪#run training

    if it % 10 is 0: #log training metrics
      tf.print('iter: ',it, ', loss: {:.3f}, acc: {:.3f}'.format(loss, acc))
      tloss.append(loss)
      tacc.append(acc)

    if it % 50 is 0: #log validation metrics
      for val_image, val_label in valid:
        val_pred = model(val_image)
        val_loss = tf.reduce_mean(tf.losses.categorical_crossentropy(tf.
 ↪one_hot(val_label,depth=2) , val_pred))
        val_acc = tf.reduce_mean(tf.metrics.categorical_accuracy(tf.
 ↪one_hot(val_label,depth=2) , val_pred))
        tf.print('iter: ',it, ', validation loss: {:.3f}, validation acc: {:.
 ↪3f}'.format(val_loss, val_acc))
        vloss.append(val_loss)
        vacc.append(val_acc)
```

```
iter:  0 , loss: 0.889, acc: 0.438
iter:  0 , validation loss: 0.794, validation acc: 0.500
iter:  10 , loss: 0.721, acc: 0.562
iter:  20 , loss: 0.710, acc: 0.594
iter:  30 , loss: 0.650, acc: 0.625
iter:  40 , loss: 0.869, acc: 0.438
```

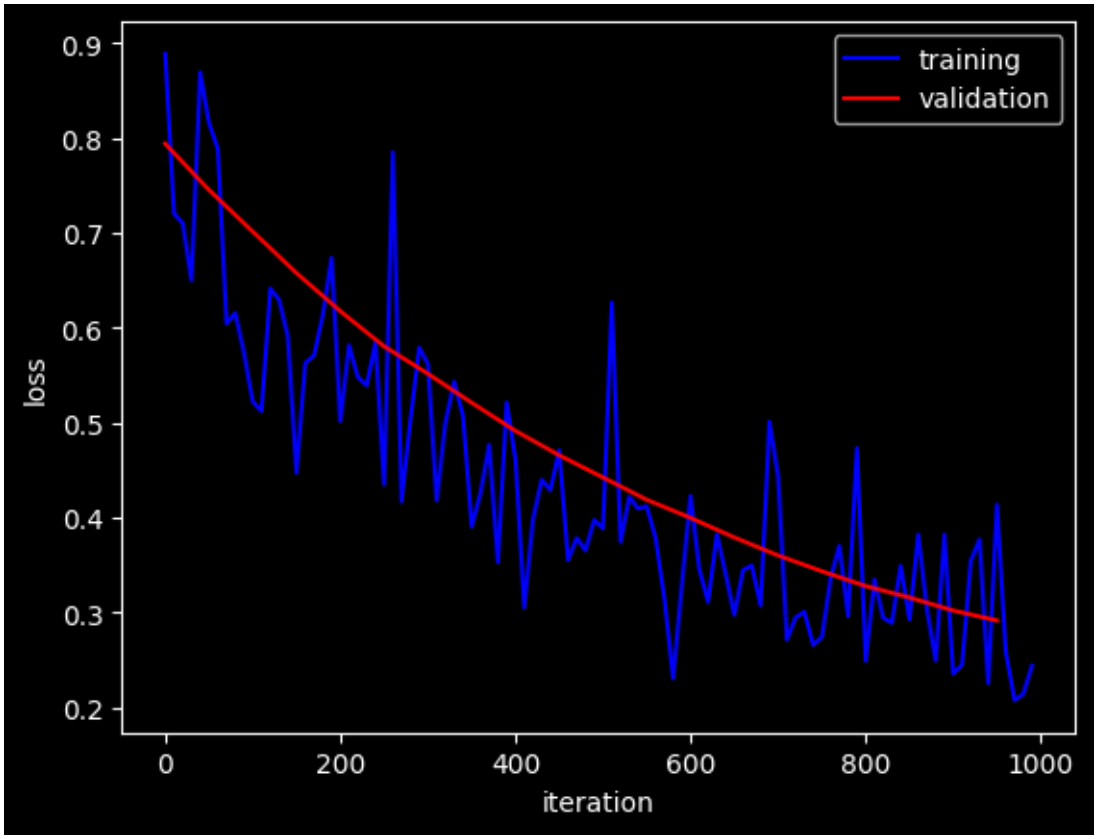
```
iter:  50 , loss: 0.816, acc: 0.562
iter:  50 , validation loss: 0.745, validation acc: 0.531
iter:  60 , loss: 0.789, acc: 0.500
iter:  70 , loss: 0.605, acc: 0.656
iter:  80 , loss: 0.616, acc: 0.719
iter:  90 , loss: 0.574, acc: 0.750
iter:  100 , loss: 0.522, acc: 0.750
iter:  100 , validation loss: 0.701, validation acc: 0.617
iter:  110 , loss: 0.512, acc: 0.719
iter:  120 , loss: 0.641, acc: 0.625
iter:  130 , loss: 0.630, acc: 0.531
iter:  140 , loss: 0.591, acc: 0.594
iter:  150 , loss: 0.447, acc: 0.812
iter:  150 , validation loss: 0.658, validation acc: 0.641
iter:  160 , loss: 0.563, acc: 0.688
iter:  170 , loss: 0.571, acc: 0.688
iter:  180 , loss: 0.614, acc: 0.719
iter:  190 , loss: 0.674, acc: 0.625
iter:  200 , loss: 0.502, acc: 0.781
iter:  200 , validation loss: 0.618, validation acc: 0.680
iter:  210 , loss: 0.582, acc: 0.688
iter:  220 , loss: 0.548, acc: 0.750
iter:  230 , loss: 0.539, acc: 0.625
iter:  240 , loss: 0.584, acc: 0.656
iter:  250 , loss: 0.435, acc: 0.812
iter:  250 , validation loss: 0.581, validation acc: 0.711
iter:  260 , loss: 0.785, acc: 0.625
iter:  270 , loss: 0.417, acc: 0.844
iter:  280 , loss: 0.502, acc: 0.812
iter:  290 , loss: 0.579, acc: 0.625
iter:  300 , loss: 0.562, acc: 0.750
iter:  300 , validation loss: 0.552, validation acc: 0.742
iter:  310 , loss: 0.418, acc: 0.906
iter:  320 , loss: 0.500, acc: 0.812
iter:  330 , loss: 0.544, acc: 0.719
iter:  340 , loss: 0.508, acc: 0.781
iter:  350 , loss: 0.391, acc: 0.906
iter:  350 , validation loss: 0.521, validation acc: 0.742
iter:  360 , loss: 0.426, acc: 0.844
iter:  370 , loss: 0.477, acc: 0.875
iter:  380 , loss: 0.353, acc: 0.906
iter:  390 , loss: 0.521, acc: 0.781
iter:  400 , loss: 0.461, acc: 0.844
iter:  400 , validation loss: 0.492, validation acc: 0.781
iter:  410 , loss: 0.305, acc: 0.938
iter:  420 , loss: 0.398, acc: 0.906
iter:  430 , loss: 0.440, acc: 0.875
iter:  440 , loss: 0.429, acc: 0.875
```

```
iter:  450 , loss: 0.471, acc: 0.781
iter:  450 , validation loss: 0.466, validation acc: 0.797
iter:  460 , loss: 0.355, acc: 0.969
iter:  470 , loss: 0.379, acc: 0.906
iter:  480 , loss: 0.366, acc: 0.938
iter:  490 , loss: 0.398, acc: 0.844
iter:  500 , loss: 0.389, acc: 0.906
iter:  500 , validation loss: 0.442, validation acc: 0.820
iter:  510 , loss: 0.627, acc: 0.750
iter:  520 , loss: 0.374, acc: 0.875
iter:  530 , loss: 0.422, acc: 0.812
iter:  540 , loss: 0.410, acc: 0.844
iter:  550 , loss: 0.412, acc: 0.906
iter:  550 , validation loss: 0.419, validation acc: 0.852
iter:  560 , loss: 0.379, acc: 0.875
iter:  570 , loss: 0.315, acc: 0.969
iter:  580 , loss: 0.231, acc: 0.969
iter:  590 , loss: 0.329, acc: 0.938
iter:  600 , loss: 0.423, acc: 0.781
iter:  600 , validation loss: 0.400, validation acc: 0.852
iter:  610 , loss: 0.346, acc: 0.906
iter:  620 , loss: 0.311, acc: 0.906
iter:  630 , loss: 0.383, acc: 0.875
iter:  640 , loss: 0.341, acc: 0.875
iter:  650 , loss: 0.298, acc: 0.906
iter:  650 , validation loss: 0.379, validation acc: 0.867
iter:  660 , loss: 0.345, acc: 0.844
iter:  670 , loss: 0.350, acc: 0.906
iter:  680 , loss: 0.307, acc: 0.938
iter:  690 , loss: 0.501, acc: 0.812
iter:  700 , loss: 0.443, acc: 0.875
iter:  700 , validation loss: 0.360, validation acc: 0.875
iter:  710 , loss: 0.271, acc: 0.938
iter:  720 , loss: 0.295, acc: 0.938
iter:  730 , loss: 0.301, acc: 0.938
iter:  740 , loss: 0.266, acc: 0.906
iter:  750 , loss: 0.274, acc: 0.938
iter:  750 , validation loss: 0.344, validation acc: 0.883
iter:  760 , loss: 0.336, acc: 0.938
iter:  770 , loss: 0.371, acc: 0.875
iter:  780 , loss: 0.296, acc: 0.938
iter:  790 , loss: 0.473, acc: 0.844
iter:  800 , loss: 0.249, acc: 0.906
iter:  800 , validation loss: 0.328, validation acc: 0.883
iter:  810 , loss: 0.335, acc: 0.875
iter:  820 , loss: 0.295, acc: 0.938
iter:  830 , loss: 0.289, acc: 0.938
iter:  840 , loss: 0.349, acc: 0.969
```

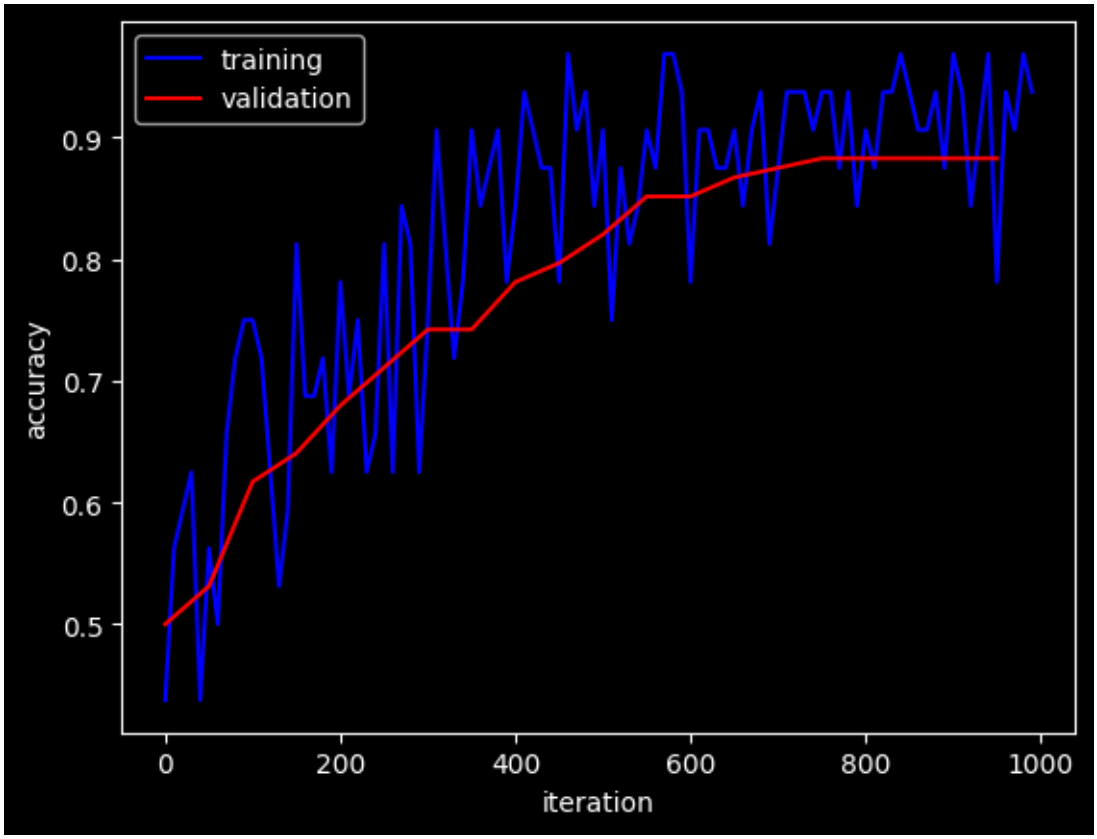
```
iter:  850 , loss: 0.293, acc: 0.938
iter:  850 , validation loss: 0.316, validation acc: 0.883
iter:  860 , loss: 0.382, acc: 0.906
iter:  870 , loss: 0.303, acc: 0.906
iter:  880 , loss: 0.249, acc: 0.938
iter:  890 , loss: 0.382, acc: 0.875
iter:  900 , loss: 0.236, acc: 0.969
iter:  900 , validation loss: 0.302, validation acc: 0.883
iter:  910 , loss: 0.245, acc: 0.938
iter:  920 , loss: 0.354, acc: 0.844
iter:  930 , loss: 0.377, acc: 0.906
iter:  940 , loss: 0.226, acc: 0.969
iter:  950 , loss: 0.414, acc: 0.781
iter:  950 , validation loss: 0.292, validation acc: 0.883
iter:  960 , loss: 0.259, acc: 0.938
iter:  970 , loss: 0.208, acc: 0.906
iter:  980 , loss: 0.214, acc: 0.969
iter:  990 , loss: 0.244, acc: 0.938
```

[98]:
```
# plot training and validation loss as a function of iteration
titers = 10*np.arange(np.shape(tloss)[0])
viters = 50*np.arange(np.shape(vloss)[0])

plt.plot(titers, tloss, c='b', label='training');
plt.plot(viters, vloss, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('loss');
```



[99]:
```
plt.plot(titers, tacc, c='b', label='training');
plt.plot(viters, vacc, c='r', label='validation');
plt.legend();
plt.xlabel('iteration');
plt.ylabel('accuracy');
```


training
validation
accuracy
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
iteration