Assignment 2

June 8, 2021

1 Assignment 2

1.0.1 Claudio Maggioni

[2]:
```
import os
import pickle
import urllib.request as http
from zipfile import ZipFile

import tensorflow as tf
import numpy as np
from PIL import Image

from tensorflow.keras import layers as keras_layers
from tensorflow.keras import backend as K
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import save_model, load_model


def load_cifar10(num_classes=3):
    """
    Downloads CIFAR-10 dataset, which already contains a training and test set,
    and return the first `num_classes` classes.
    Example of usage:

    >>> (x_train, y_train), (x_test, y_test) = load_cifar10()

    :param num_classes: int, default is 3 as required by the assignment.
    :return: the filtered data.
    """
    (x_train_all, y_train_all), (x_test_all, y_test_all) = cifar10.load_data()

    fil_train = tf.where(y_train_all[:, 0] < num_classes)[:, 0]
    fil_test = tf.where(y_test_all[:, 0] < num_classes)[:, 0]

    y_train = y_train_all[fil_train]
    y_test = y_test_all[fil_test]
```

```
    x_train = x_train_all[fil_train]
    x_test = x_test_all[fil_test]

    return (x_train, y_train), (x_test, y_test)


def load_rps(download=False, path='rps', reduction_factor=1):
    """
    Downloads the rps dataset and returns the training and test sets.
    Example of usage:

    >>> (x_train, y_train), (x_test, y_test) = load_rps()

    :param download: bool, default is False but for the first call should be␣
↪True.
    :param path: str, subdirectory in which the images should be downloaded,␣
↪default is 'rps'.
    :param reduction_factor: int, factor of reduction of the dataset (len =␣
↪old_len // reduction_factor).
    :return: the images and labels split into training and validation sets.
    """
    url = 'https://drive.switch.ch/index.php/s/xjXhuYDUzoZvL02/download'
    classes = ('rock', 'paper', 'scissors')
    rps_dir = os.path.abspath(path)
    filename = os.path.join(rps_dir, 'data.zip')
    if not os.path.exists(rps_dir) and not download:
        raise ValueError("Dataset not in the path. You should call this␣
↪function with `download=True` the first time.")
    if download:
        os.makedirs(rps_dir, exist_ok=True)
        print(f"Downloading rps images in {rps_dir} (may take a couple of␣
↪minutes)")
        path, msg = http.urlretrieve(url, filename)
        with ZipFile(path, 'r') as zip_ref:
            zip_ref.extractall(rps_dir)
        os.remove(filename)
    train_dir, test_dir = os.path.join(rps_dir, 'train'), os.path.join(rps_dir,␣
↪'test')
    print("Loading training set...")
    x_train, y_train = load_images_with_label(train_dir, classes)
    x_train, y_train = x_train[::reduction_factor], y_train[::reduction_factor]
    print("Loaded %d images for training" % len(y_train))
    print("Loading test set...")
    x_test, y_test = load_images_with_label(test_dir, classes)
    x_test, y_test = x_test[::reduction_factor], y_test[::reduction_factor]
    print("Loaded %d images for testing" % len(y_test))
```

```
    return (x_train, y_train), (x_test, y_test)


def make_dataset(imgs, labels, label_map, img_size, rgb=True, keepdim=True,␣
↪shuffle=True):
    x = []
    y = []
    n_classes = len(list(label_map.keys()))
    for im, l in zip(imgs, labels):
        # preprocess img
        x_i = im.resize(img_size)
        if not rgb:
            x_i = x_i.convert('L')
        x_i = np.asarray(x_i)
        if not keepdim:
            x_i = x_i.reshape(-1)

        # encode label
        y_i = np.zeros(n_classes)
        y_i[label_map[l]] = 1.

        x.append(x_i)
        y.append(y_i)
    x, y = np.array(x).astype('float32'), np.array(y)
    if shuffle:
        idxs = np.arange(len(y))
        np.random.shuffle(idxs)
        x, y = x[idxs], y[idxs]
    return x, y


def load_images(path):
    img_files = os.listdir(path)
    imgs, labels = [], []
    for i in img_files:
        if i.endswith('.jpg'):
            # load the image (here you might want to resize the img to save␣
↪memory)
            imgs.append(Image.open(os.path.join(path, i)).copy())
    return imgs


def load_images_with_label(path, classes):
    imgs, labels = [], []
    for c in classes:
        # iterate over all the files in the folder
        c_imgs = load_images(os.path.join(path, c))
```

```
        imgs.extend(c_imgs)
        labels.extend([c] * len(c_imgs))
    return imgs, labels


def save_keras_model(model, filename):
    """
    Saves a Keras model to disk.
    Example of usage:

    >>> model = Sequential()
    >>> model.add(Dense(...))
    >>> model.compile(...)
    >>> model.fit(...)
    >>> save_keras_model(model, 'my_model.h5')

    :param model: the model to save;
    :param filename: string, path to the file in which to store the model.
    :return: the model.
    """
    save_model(model, filename)


def load_keras_model(filename):
    """
    Loads a compiled Keras model saved with models.save_model.

    :param filename: string, path to the file storing the model.
    :return: the model.
    """
    model = load_model(filename)
    return model


def save_vgg16(model, filename='nn_task2.pkl', additional_args=()):
    """
    Optimize task2 model by only saving the layers after vgg16. This function
    assumes that you only added Flatten and Dense layers. If it is not the case,
    you should include into `additional_args` other layers' attributes you
    need.

    :param filename: string, path to the file in which to store the model.
    :param additional_args: tuple or list, additional layers' attributes to be
    saved. Default are ['units', 'activation', 'use_bias']
    :return: the path of the saved model.
    """
    filename = filename if filename.endswith('.pkl') else (filename + '.pkl')
```

```
    args = ['units', 'activation', 'use_bias', *additional_args]
    layers = []
    for l in model.layers[1:]:
        layer = dict()
        layer['class'] = l.__class__.__name__
        if l.weights:
            layer['weights'] = l.get_weights()
            layer['kwargs'] = {k: v for k, v in vars(l).items() if k in args}
        layers.append(layer)

    with open(filename, 'wb') as fp:
        pickle.dump(layers, fp)

    return os.path.abspath(filename)


def load_vgg16(filename='nn_task2.pkl', img_h=224, img_w=224):
    """
    Loads the model saved with save_vgg16.

    :param filename: string, path to the file storing the model.
    :param img_h: int, the height of the input image.
    :param img_w: int, the width of the input image.
    :return: the model.
    """
    K.clear_session()

    vgg16 = applications.VGG16(weights='imagenet',
                               include_top=False,
                               input_shape=(img_h, img_w, 3))
    model = Sequential()
    model.add(vgg16)

    with open(filename, 'rb') as fp:
        layers = pickle.load(fp)
    for l in layers:
        cls = getattr(keras_layers, l['class'])
        if 'weights' in l:
            layer = cls(**l['kwargs'])
            model.add(layer)
            model.layers[-1].set_weights(l['weights'])
        else:
            model.add(cls())

    model.trainable = False
    return model
```

2 Exercise 1

[3]:
```
# Load the training and test CIFAR10 data
(x_train, y_train), (x_test, y_test) = load_cifar10()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
```

[ ]:
```
# Normalize the train and test data
x_train_n = x_train / 255
x_test_n = x_test / 255

# Check if only 3 classes were loaded (no output should be printed)
for e in y_train:
  if e[0] not in [0,1,2]:
    print(e[0])
```

[ ]:
```
from tensorflow.keras import utils

n_classes = 3

# Convert output data to one-hot encoding
y_train_n = utils.to_categorical(y_train, n_classes)
y_test_n = utils.to_categorical(y_test, n_classes)
```

[ ]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, \
  Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping, CSVLogger

# Build the CIFAR10 model architecture
model = Sequential()
model.add(Conv2D(8, (5, 5), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (3, 3), strides=(2,2), activation='relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(8, activation='tanh'))
model.add(Dense(n_classes, activation='softmax'))

# Compile the model and print model architecture
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.003),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
# Implement early stopping monitoring validation accuracy
callback = EarlyStopping(monitor='val_accuracy',
                  patience=10,
                  restore_best_weights=True)

# Log training data in the indicated CSV file
log_task1 = CSVLogger('my_civar10.csv')

# Train the model
batch_size = 128
epochs = 500
model.fit(x_train_n,
          y_train_n,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2,
          callbacks=[callback, log_task1])
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 8)         608
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 6, 6, 16)          1168
_________________________________________________________________
average_pooling2d (AveragePo (None, 3, 3, 16)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 144)               0
_________________________________________________________________
dense_13 (Dense)             (None, 8)                 1160
_________________________________________________________________
dense_14 (Dense)             (None, 3)                 27
=================================================================
Total params: 2,963
Trainable params: 2,963
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
94/94 [==============================] - 1s 6ms/step - loss: 0.9172 - accuracy:
0.5718 - val_loss: 0.9488 - val_accuracy: 0.5113
Epoch 2/500
94/94 [==============================] - 0s 5ms/step - loss: 0.7498 - accuracy:
0.6758 - val_loss: 0.6664 - val_accuracy: 0.7343
Epoch 3/500
94/94 [==============================] - 0s 5ms/step - loss: 0.6778 - accuracy:
```

```
0.7107 - val_loss: 0.6071 - val_accuracy: 0.7557
Epoch 4/500
94/94 [==============================] - 0s 5ms/step - loss: 0.6342 - accuracy:
0.7352 - val_loss: 0.6572 - val_accuracy: 0.7210
Epoch 5/500
94/94 [==============================] - 0s 5ms/step - loss: 0.5902 - accuracy:
0.7538 - val_loss: 0.5681 - val_accuracy: 0.7710
Epoch 6/500
94/94 [==============================] - 0s 5ms/step - loss: 0.5556 - accuracy:
0.7712 - val_loss: 0.5319 - val_accuracy: 0.7933
Epoch 7/500
94/94 [==============================] - 0s 4ms/step - loss: 0.5265 - accuracy:
0.7832 - val_loss: 0.4979 - val_accuracy: 0.8007
Epoch 8/500
94/94 [==============================] - 0s 5ms/step - loss: 0.5014 - accuracy:
0.7976 - val_loss: 0.4851 - val_accuracy: 0.8007
Epoch 9/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4874 - accuracy:
0.8020 - val_loss: 0.4577 - val_accuracy: 0.8227
Epoch 10/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4745 - accuracy:
0.8067 - val_loss: 0.4667 - val_accuracy: 0.8150
Epoch 11/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4576 - accuracy:
0.8162 - val_loss: 0.4599 - val_accuracy: 0.8180
Epoch 12/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4456 - accuracy:
0.8223 - val_loss: 0.4813 - val_accuracy: 0.8133
Epoch 13/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4402 - accuracy:
0.8247 - val_loss: 0.4650 - val_accuracy: 0.8140
Epoch 14/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4313 - accuracy:
0.8288 - val_loss: 0.4428 - val_accuracy: 0.8280
Epoch 15/500
94/94 [==============================] - 0s 5ms/step - loss: 0.4172 - accuracy:
0.8344 - val_loss: 0.4678 - val_accuracy: 0.8157
Epoch 16/500
94/94 [==============================] - 0s 4ms/step - loss: 0.4134 - accuracy:
0.8351 - val_loss: 0.4249 - val_accuracy: 0.8353
Epoch 17/500
94/94 [==============================] - 1s 5ms/step - loss: 0.4058 - accuracy:
0.8365 - val_loss: 0.4124 - val_accuracy: 0.8430
Epoch 18/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3955 - accuracy:
0.8430 - val_loss: 0.4277 - val_accuracy: 0.8313
Epoch 19/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3878 - accuracy:
```

```
0.8466 - val_loss: 0.4089 - val_accuracy: 0.8367
Epoch 20/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3865 - accuracy:
0.8499 - val_loss: 0.5154 - val_accuracy: 0.7947
Epoch 21/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3827 - accuracy:
0.8513 - val_loss: 0.4198 - val_accuracy: 0.8393
Epoch 22/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3752 - accuracy:
0.8518 - val_loss: 0.4005 - val_accuracy: 0.8447
Epoch 23/500
94/94 [==============================] - 0s 4ms/step - loss: 0.3663 - accuracy:
0.8551 - val_loss: 0.4908 - val_accuracy: 0.8120
Epoch 24/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3641 - accuracy:
0.8571 - val_loss: 0.4103 - val_accuracy: 0.8430
Epoch 25/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3591 - accuracy:
0.8580 - val_loss: 0.3885 - val_accuracy: 0.8547
Epoch 26/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3537 - accuracy:
0.8607 - val_loss: 0.4419 - val_accuracy: 0.8363
Epoch 27/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3477 - accuracy:
0.8650 - val_loss: 0.3892 - val_accuracy: 0.8490
Epoch 28/500
94/94 [==============================] - 0s 4ms/step - loss: 0.3448 - accuracy:
0.8679 - val_loss: 0.4389 - val_accuracy: 0.8340
Epoch 29/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3396 - accuracy:
0.8660 - val_loss: 0.3977 - val_accuracy: 0.8500
Epoch 30/500
94/94 [==============================] - 0s 4ms/step - loss: 0.3386 - accuracy:
0.8673 - val_loss: 0.4488 - val_accuracy: 0.8283
Epoch 31/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3359 - accuracy:
0.8698 - val_loss: 0.3940 - val_accuracy: 0.8473
Epoch 32/500
94/94 [==============================] - 0s 4ms/step - loss: 0.3290 - accuracy:
0.8699 - val_loss: 0.4283 - val_accuracy: 0.8343
Epoch 33/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3222 - accuracy:
0.8747 - val_loss: 0.3796 - val_accuracy: 0.8540
Epoch 34/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3185 - accuracy:
0.8778 - val_loss: 0.3872 - val_accuracy: 0.8553
Epoch 35/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3212 - accuracy:
```

```
0.8730 - val_loss: 0.4123 - val_accuracy: 0.8397
Epoch 36/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3160 - accuracy:
0.8738 - val_loss: 0.3843 - val_accuracy: 0.8533
Epoch 37/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3122 - accuracy:
0.8777 - val_loss: 0.3735 - val_accuracy: 0.8587
Epoch 38/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3127 - accuracy:
0.8755 - val_loss: 0.3968 - val_accuracy: 0.8450
Epoch 39/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3069 - accuracy:
0.8798 - val_loss: 0.4014 - val_accuracy: 0.8417
Epoch 40/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3056 - accuracy:
0.8823 - val_loss: 0.4605 - val_accuracy: 0.8237
Epoch 41/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3014 - accuracy:
0.8811 - val_loss: 0.3749 - val_accuracy: 0.8650
Epoch 42/500
94/94 [==============================] - 0s 5ms/step - loss: 0.3015 - accuracy:
0.8819 - val_loss: 0.3943 - val_accuracy: 0.8593
Epoch 43/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2996 - accuracy:
0.8835 - val_loss: 0.4209 - val_accuracy: 0.8373
Epoch 44/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2928 - accuracy:
0.8851 - val_loss: 0.3775 - val_accuracy: 0.8687
Epoch 45/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2901 - accuracy:
0.8882 - val_loss: 0.4303 - val_accuracy: 0.8313
Epoch 46/500
94/94 [==============================] - 0s 4ms/step - loss: 0.2915 - accuracy:
0.8869 - val_loss: 0.4693 - val_accuracy: 0.8233
Epoch 47/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2878 - accuracy:
0.8877 - val_loss: 0.3719 - val_accuracy: 0.8610
Epoch 48/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2852 - accuracy:
0.8861 - val_loss: 0.3751 - val_accuracy: 0.8647
Epoch 49/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2818 - accuracy:
0.8891 - val_loss: 0.4302 - val_accuracy: 0.8427
Epoch 50/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2842 - accuracy:
0.8893 - val_loss: 0.4167 - val_accuracy: 0.8477
Epoch 51/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2788 - accuracy:
```

```
0.8915 - val_loss: 0.4626 - val_accuracy: 0.8300
Epoch 52/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2767 - accuracy:
0.8950 - val_loss: 0.4610 - val_accuracy: 0.8333
Epoch 53/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2742 - accuracy:
0.8938 - val_loss: 0.3817 - val_accuracy: 0.8633
Epoch 54/500
94/94 [==============================] - 0s 5ms/step - loss: 0.2761 - accuracy:
0.8933 - val_loss: 0.4188 - val_accuracy: 0.8477
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f9efe11e110>
```

2.0.1 Plot validation loss and accuracy curves

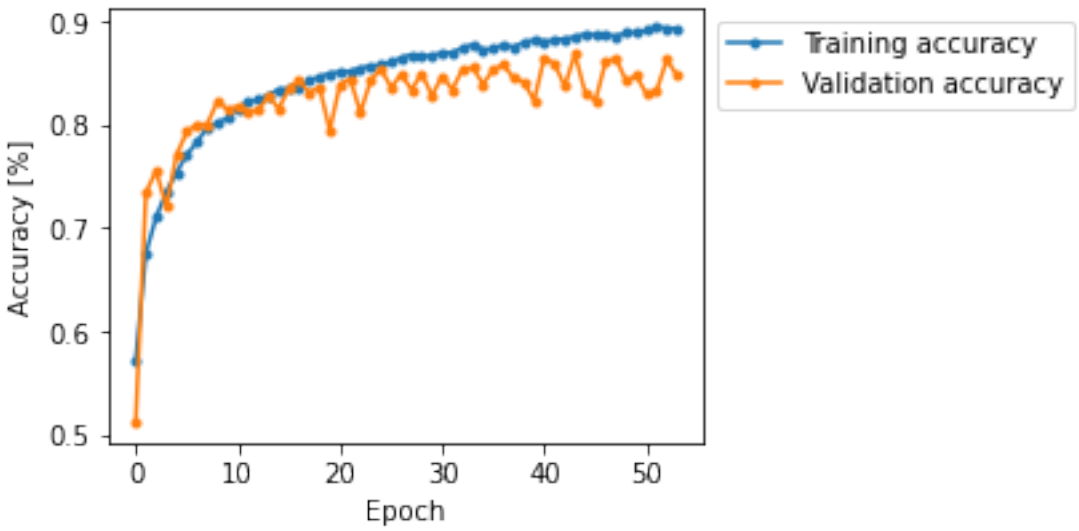
[ ]:
```
import pandas as pd

# Load the CSV with saved training data
df = pd.read_csv('my_civar10.csv')
print(df[df.epoch == 0])
```

```
   epoch  accuracy      loss  val_accuracy  val_loss
0      0  0.571833  0.917225      0.511333  0.948794
```

[ ]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

# Plot training and validation accuracy w.r.t. epoch for CIFAR10 model
plt.figure(figsize=(4,3))
ax = plt.gca()
lines = []
FEATURES = ['accuracy', 'val_accuracy']
for feature in FEATURES:
  lines.append(ax.plot(df["epoch"], df[feature], marker='.')[0])
plt.xlabel("Epoch")
plt.ylabel("Accuracy [%]")
lgd = plt.legend(lines, ["Training accuracy", "Validation accuracy"],
                 loc="best", bbox_to_anchor=(1,1))
plt.show()
```



2.0.2 Statistical tests on CIFAR classifier

[ ]:
```
# Compute loss and accuracy on the test set
print(np.shape(y_test_n))
test_loss, test_accuracy = model.evaluate(x_test_n, y_test_n)
print("Test accuracy is: %g" % test_accuracy)
```

```
(3000, 3)
94/94 [==============================] - 0s 2ms/step - loss: 0.3649 - accuracy:
0.8640
Test accuracy is: 0.864
```

[16]:
```
# Compute confidence interval for accuracy using binomial distribution
import numpy as np
import scipy.stats as st
from statsmodels.stats.proportion import proportion_confint

test_accuracy = 0.864
n = len(y_test)

proportion_confint(test_accuracy * n, n, method='binom_test', alpha=0.05)
```

[16]:
```
(0.8512018385349547, 0.8758694331900033)
```

2.0.3 Save the model

[ ]:
```
# Save the model
save_keras_model(model, filename='/content/nn_task1.h5')
```

```
None
```

3 Exercise 2

[6]:
```
# Load RPS data
(x_train, y_train), (x_test, y_test) = load_rps(download=True)
```

```
Downloading rps images in /content/rps (may take a couple of minutes)
Loading training set…
Loaded 1500 images for training
Loading test set…
Loaded 300 images for testing
```

[7]:
```
import torch
import tensorflow as tf
from tensorflow.keras import applications
from keras.preprocessing.image import img_to_array, array_to_img

# Resize the input images and normalize them according to VGG16 normalization
# factors
def process_vgg16(x):
  x_n = [e.resize((224,224)) for e in x]
  for i in range(len(x)):
    bgr = img_to_array(x_n[i])[..., ::-1]
    mean = [103.939, 116.779, 123.68]
    bgr -= mean
    x_n[i] = bgr
  return x_n

# Process train and test set
x_train_n = tf.convert_to_tensor(process_vgg16(x_train))
x_test_n = tf.convert_to_tensor(process_vgg16(x_test))
```

[8]:
```
from tensorflow.keras import utils

LABELS = set(y_train)
MAP = {'scissors': 0, 'paper': 1, 'rock': 2}
print(MAP)

# Convert string labels to numerical ones according to MAP
mapfunc = np.vectorize(lambda x: MAP[x])

# Convert numerical labels to one-hot encoding
y_train_n = utils.to_categorical(mapfunc(y_train), len(LABELS))
y_test_n = utils.to_categorical(mapfunc(y_test), len(LABELS))
```

```
{'scissors': 0, 'paper': 1, 'rock': 2}
```

[9]:
```
# Download VGG16 convolution weights and architecture
```

```
from tensorflow.keras import Sequential, optimizers, applications
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping, CSVLogger

# Build the VGG16 network and download pre-trained weights and remove the last
# dense layers.
vgg16 = applications.VGG16(weights='imagenet',
                           include_top=False,
                           input_shape=(224, 224, 3))

# Freezes the network weights
vgg16.trainable = False
```

[ ]:
```
# Build VGG16-based classifier
net = Sequential()
net.add(vgg16)
net.add(Flatten())
net.add(Dense(128, activation='relu'))
net.add(Dropout(0.25))
net.add(Dense(3, activation='softmax'))

# Compile and print network architecture
net.compile(optimizer=optimizers.Adam(learning_rate=0.001),
            loss='categorical_crossentropy',
            metrics=['acc'])
net.summary()

# Implement early stopping monitoring validation loss
es = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Save training loss and accuracy over epochs in indicated CSV
log_task2_noaug = CSVLogger('my_vgg16_noaug.csv')

# Fit the model with not data augmented training and validation data
history = net.fit(x_train_n,
                  y_train_n,
                  batch_size=8,
                  epochs=50,
                  validation_split=0.2,
                  verbose=1,
                  callbacks=[es, log_task2_noaug])
```

[ ]:
```
# Evaluate non-data-augmented model on test set
scores = net.evaluate(x_test_n, y_test_n)
print('Test loss: {} - Accuracy: {}'.format(*scores))
```

```
# Save the model
save_keras_model(net, '/content/ynn_task2_noaug.h5')
```

```
10/10 [==============================] - 1s 118ms/step - loss: 0.5268 - acc:
0.7733
Test loss: 0.5268241763114929 - Accuracy: 0.7733333110809326
None
```

[10]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_gen = ImageDataGenerator(width_shift_range=0.15,    # horizontal␣
 ↪translation
                               height_shift_range=0.15,   # vertical translation
                               channel_shift_range=0.3,   # random channel␣
 ↪shifts
                               rotation_range=360,        # rotation
                               zoom_range=0.3,            # zoom in/out randomly
                               shear_range=15,            # deformation
                              )
val_gen = ImageDataGenerator()

# Generate data-augmented training and validation set
train_loader = train_gen.flow(x_train_n, y_train_n, batch_size=40)
validation_loader = train_gen.flow(x_train_n, y_train_n, batch_size=10)
```

[12]:
```
# Build VGG16-based classifier
net2 = Sequential()
net2.add(vgg16)
net2.add(Flatten())
net2.add(Dense(128, activation='relu'))
net2.add(Dropout(0.25))
net2.add(Dense(3, activation='softmax'))

# Implement early stopping monitoring validation loss
es2 = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Compile and print network architecture
net2.compile(optimizer=optimizers.Adam(learning_rate=0.001),
             loss='categorical_crossentropy',
             metrics=['acc'])
net2.summary()

# Save training loss and accuracy over epochs in indicated CSV
log_aug = CSVLogger('my_vgg16_aug.csv')

# Fit the model with data augmented training and validation data
history = net2.fit(train_loader,
```

```
                  batch_size=16,
                  epochs=50,
                  validation_data=validation_loader,
                  verbose=1,
                  callbacks=[es2, log_aug])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3211392
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 387
=================================================================
Total params: 17,926,467
Trainable params: 3,211,779
Non-trainable params: 14,714,688
_________________________________________________________________
Epoch 1/50
38/38 [==============================] - 48s 1s/step - loss: 5.9368 - acc:
0.4780 - val_loss: 0.9557 - val_acc: 0.4833
Epoch 2/50
38/38 [==============================] - 32s 842ms/step - loss: 0.9711 - acc:
0.5707 - val_loss: 0.9521 - val_acc: 0.5767
Epoch 3/50
38/38 [==============================] - 32s 842ms/step - loss: 0.9213 - acc:
0.6033 - val_loss: 0.7592 - val_acc: 0.6940
Epoch 4/50
38/38 [==============================] - 32s 839ms/step - loss: 0.8037 - acc:
0.6560 - val_loss: 0.6932 - val_acc: 0.7393
Epoch 5/50
38/38 [==============================] - 32s 845ms/step - loss: 0.8075 - acc:
0.6547 - val_loss: 0.6627 - val_acc: 0.7313
Epoch 6/50
38/38 [==============================] - 33s 868ms/step - loss: 0.7375 - acc:
0.6980 - val_loss: 0.6030 - val_acc: 0.7540
Epoch 7/50
38/38 [==============================] - 33s 874ms/step - loss: 0.7086 - acc:
0.7080 - val_loss: 0.5946 - val_acc: 0.7680
Epoch 8/50
38/38 [==============================] - 32s 847ms/step - loss: 0.6479 - acc:
0.7100 - val_loss: 0.5195 - val_acc: 0.8107
```

```
Epoch 9/50
38/38 [==============================] - 32s 844ms/step - loss: 0.6702 - acc:
0.7167 - val_loss: 0.5435 - val_acc: 0.7847
Epoch 10/50
38/38 [==============================] - 32s 849ms/step - loss: 0.6077 - acc:
0.7500 - val_loss: 0.4991 - val_acc: 0.8180
Epoch 11/50
38/38 [==============================] - 32s 843ms/step - loss: 0.6208 - acc:
0.7333 - val_loss: 0.4954 - val_acc: 0.8060
Epoch 12/50
38/38 [==============================] - 32s 851ms/step - loss: 0.6244 - acc:
0.7313 - val_loss: 0.5026 - val_acc: 0.8060
Epoch 13/50
38/38 [==============================] - 32s 848ms/step - loss: 0.5998 - acc:
0.7640 - val_loss: 0.4789 - val_acc: 0.8127
Epoch 14/50
38/38 [==============================] - 32s 839ms/step - loss: 0.5802 - acc:
0.7507 - val_loss: 0.4533 - val_acc: 0.8273
Epoch 15/50
38/38 [==============================] - 32s 846ms/step - loss: 0.5767 - acc:
0.7533 - val_loss: 0.4746 - val_acc: 0.8233
Epoch 16/50
38/38 [==============================] - 32s 839ms/step - loss: 0.5643 - acc:
0.7600 - val_loss: 0.4329 - val_acc: 0.8253
Epoch 17/50
38/38 [==============================] - 32s 852ms/step - loss: 0.5584 - acc:
0.7673 - val_loss: 0.4671 - val_acc: 0.8067
Epoch 18/50
38/38 [==============================] - 32s 850ms/step - loss: 0.5940 - acc:
0.7587 - val_loss: 0.4413 - val_acc: 0.8300
Epoch 19/50
38/38 [==============================] - 32s 844ms/step - loss: 0.5850 - acc:
0.7573 - val_loss: 0.4237 - val_acc: 0.8373
Epoch 20/50
38/38 [==============================] - 32s 841ms/step - loss: 0.5519 - acc:
0.7820 - val_loss: 0.4195 - val_acc: 0.8373
Epoch 21/50
38/38 [==============================] - 32s 845ms/step - loss: 0.5243 - acc:
0.7867 - val_loss: 0.4205 - val_acc: 0.8433
Epoch 22/50
38/38 [==============================] - 32s 840ms/step - loss: 0.5330 - acc:
0.7800 - val_loss: 0.4232 - val_acc: 0.8427
Epoch 23/50
38/38 [==============================] - 32s 848ms/step - loss: 0.5486 - acc:
0.7927 - val_loss: 0.4149 - val_acc: 0.8393
Epoch 24/50
38/38 [==============================] - 32s 843ms/step - loss: 0.5066 - acc:
0.7987 - val_loss: 0.4016 - val_acc: 0.8480
```

```
Epoch 25/50
38/38 [==============================] - 32s 838ms/step - loss: 0.5062 - acc:
0.8000 - val_loss: 0.4163 - val_acc: 0.8360
Epoch 26/50
38/38 [==============================] - 31s 837ms/step - loss: 0.4952 - acc:
0.7940 - val_loss: 0.3879 - val_acc: 0.8533
Epoch 27/50
38/38 [==============================] - 31s 835ms/step - loss: 0.5135 - acc:
0.7893 - val_loss: 0.3924 - val_acc: 0.8480
Epoch 28/50
38/38 [==============================] - 31s 830ms/step - loss: 0.5359 - acc:
0.7933 - val_loss: 0.3887 - val_acc: 0.8547
Epoch 29/50
38/38 [==============================] - 32s 849ms/step - loss: 0.4884 - acc:
0.8040 - val_loss: 0.3913 - val_acc: 0.8540
Epoch 30/50
38/38 [==============================] - 32s 839ms/step - loss: 0.4803 - acc:
0.8040 - val_loss: 0.4148 - val_acc: 0.8447
Epoch 31/50
38/38 [==============================] - 31s 836ms/step - loss: 0.5072 - acc:
0.8060 - val_loss: 0.3828 - val_acc: 0.8527
Epoch 32/50
38/38 [==============================] - 31s 833ms/step - loss: 0.4988 - acc:
0.7980 - val_loss: 0.4236 - val_acc: 0.8580
Epoch 33/50
38/38 [==============================] - 32s 844ms/step - loss: 0.4721 - acc:
0.8093 - val_loss: 0.4001 - val_acc: 0.8580
Epoch 34/50
38/38 [==============================] - 31s 835ms/step - loss: 0.4841 - acc:
0.8100 - val_loss: 0.3656 - val_acc: 0.8753
Epoch 35/50
38/38 [==============================] - 31s 833ms/step - loss: 0.4884 - acc:
0.8067 - val_loss: 0.3772 - val_acc: 0.8547
Epoch 36/50
38/38 [==============================] - 31s 834ms/step - loss: 0.5038 - acc:
0.8060 - val_loss: 0.3824 - val_acc: 0.8627
Epoch 37/50
38/38 [==============================] - 32s 839ms/step - loss: 0.4988 - acc:
0.8007 - val_loss: 0.3792 - val_acc: 0.8600
Epoch 38/50
38/38 [==============================] - 31s 840ms/step - loss: 0.4807 - acc:
0.8133 - val_loss: 0.3693 - val_acc: 0.8727
Epoch 39/50
38/38 [==============================] - 31s 836ms/step - loss: 0.4792 - acc:
0.8053 - val_loss: 0.3456 - val_acc: 0.8720
Epoch 40/50
38/38 [==============================] - 32s 837ms/step - loss: 0.4598 - acc:
0.8120 - val_loss: 0.4263 - val_acc: 0.8493
```

```
Epoch 41/50
38/38 [==============================] - 31s 835ms/step - loss: 0.4712 - acc:
0.8120 - val_loss: 0.3842 - val_acc: 0.8687
Epoch 42/50
38/38 [==============================] - 31s 834ms/step - loss: 0.4660 - acc:
0.8060 - val_loss: 0.3825 - val_acc: 0.8673
Epoch 43/50
38/38 [==============================] - 31s 836ms/step - loss: 0.4617 - acc:
0.8193 - val_loss: 0.3563 - val_acc: 0.8667
Epoch 44/50
38/38 [==============================] - 32s 841ms/step - loss: 0.4561 - acc:
0.8227 - val_loss: 0.3650 - val_acc: 0.8647
Epoch 45/50
38/38 [==============================] - 31s 835ms/step - loss: 0.4573 - acc:
0.8240 - val_loss: 0.3844 - val_acc: 0.8507
Epoch 46/50
38/38 [==============================] - 32s 843ms/step - loss: 0.4898 - acc:
0.8020 - val_loss: 0.3444 - val_acc: 0.8740
Epoch 47/50
38/38 [==============================] - 31s 836ms/step - loss: 0.4352 - acc:
0.8273 - val_loss: 0.3574 - val_acc: 0.8640
Epoch 48/50
38/38 [==============================] - 32s 837ms/step - loss: 0.4651 - acc:
0.8187 - val_loss: 0.3359 - val_acc: 0.8793
Epoch 49/50
38/38 [==============================] - 32s 841ms/step - loss: 0.4681 - acc:
0.8173 - val_loss: 0.3494 - val_acc: 0.8687
Epoch 50/50
38/38 [==============================] - 31s 835ms/step - loss: 0.4628 - acc:
0.8113 - val_loss: 0.3382 - val_acc: 0.8760
```

[13]:
```
# Evaluate non-data-augmented model on test set
scores = net2.evaluate(x_test_n, y_test_n)
print('Test loss: {} - Accuracy: {}'.format(*scores))

# Save the model
print(save_keras_model(net2, filename='/content/nn_task2_aug.h5'))
```

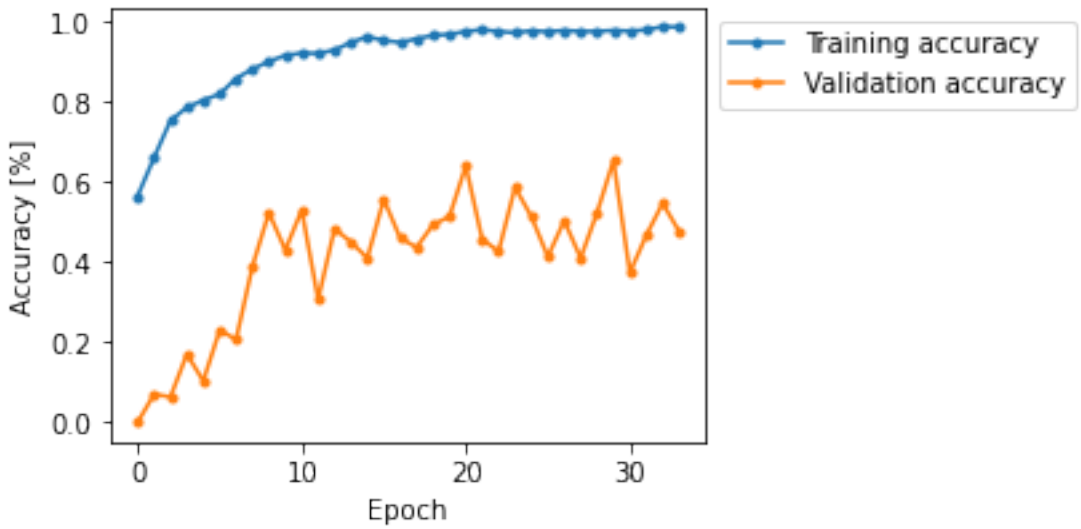
```
10/10 [==============================] - 11s 439ms/step - loss: 0.2486 - acc:
0.9000
Test loss: 0.2485782653093338 - Accuracy: 0.8999999761581421
None
```

3.0.1 Plot validation loss and accuracy curves (non-data-augmented model)

[4]:
```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt

df = pd.read_csv('my_vgg16_noaug.csv')
print(df[df.epoch == 0])
plt.figure(figsize=(4,3))
ax = plt.gca()
lines = []
FEATURES = ['acc', 'val_acc']
for feature in FEATURES:
  lines.append(ax.plot(df["epoch"],df[feature], marker='.')[0])
plt.xlabel("Epoch")
plt.ylabel("Accuracy [%]")
lgd = plt.legend(lines, ["Training accuracy", "Validation accuracy"],
                 loc="best", bbox_to_anchor=(1,1))
plt.show()
```

```
   epoch    acc      loss  val_acc  val_loss
0      0  0.565  0.955588      0.0  1.627829
```



3.0.2 Plot validation loss and accuracy curves (data augmented model)

[15]:
```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('my_vgg16_aug.csv')
print(df[df.epoch == 0])
plt.figure(figsize=(4,3))
ax = plt.gca()
lines = []
FEATURES = ['acc', 'val_acc']
for feature in FEATURES:
  lines.append(ax.plot(df["epoch"],df[feature], marker='.')[0])
plt.xlabel("Epoch")
plt.ylabel("Accuracy [%]")
lgd = plt.legend(lines, ["Training accuracy", "Validation accuracy"],
                 loc="best", bbox_to_anchor=(1,1))
plt.show()
```

```
   epoch    acc     loss   val_acc  val_loss
0      0  0.478  5.93678  0.483333  0.955677
```