

```
# Fill in your names using the format below
your_name_1 = "Booy, Harm"
your_name_2 = "Dongen, Summer van"
your_name_3 = "Groen, Maud"
your_name_4 = "Wasser, Jeroen"
your_name_5 = "Hugten, Koen van"
```

## ▼ Data Mining: Assignment 2

In this assignment you are asked to build convolutional neural networks for two quite different image datasets.

To complete this assignment, you must submit *all* your work, including the trained models (because we need to be able to test them). You will need to upload a zip file to Canvas with:

- This notebook, containing ALL your code
- A PDF report, including a download link to all other files (e.g. the trained models). Keep the report under 10 pages.

For the report you can use any text editor, but export it to PDF. The report should explain all your design decisions, answers to the questions below, analyses of your models, and a clear interpretation of your results. Make the report self-contained, and copy all necessary plots from this notebook to your report.

To submit the other files (e.g. the stored models), you can put them in a GitHub repository and reference the link in your report. Alternatively, you can submit them as a large zipped file in another way, as long as a download link is in the report.

```
# Imports and version checking
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import tensorflow as tf
from packaging import version

sklearn_version = sklearn.__version__
tensorflow_version = tf.__version__
if version.parse(tensorflow_version) < version.parse("2.2.0"):
    print("Tensorflow is outdated. This is version {}. Please update to 2.2 or later (e.g. 2.4)".format(tensorflow_version))
```

```
elif version.parse(tensorflow_version) < version.parse("2.4.0"):
    print("Tensorflow version is <2.4. This will likely work but we recommend updating to 2.4".format(tensorflow_version))
else:
    print("Looks good. You may continue :)")
```

Looks good. You may continue :)

## ▼ Storing and submitting files

The evaluation functions used in this notebook will automatically store models for you. Be sure to submit all .h5 and .p files, as well as any .json files created (these are created only for large models).

If you want to run and solve the notebook on your local machine/laptop, fill in the path 'base\_dir' to your assignment folder into the next cell.

If you use Colab, we recommend that you link it to your Google Drive by uncommenting the code below.

- Create an 'assignment' folder in your Google Drive with this notebook
- Uncomment and run the code below to give permissions to store the model files
- Fill in the path to your assignment folder below

- It's likely `base_dir = '/content/drive/My Drive/assignment'`

```
# For storing the files in a local folder.
base_dir = 'C:/Users/20182440/Documents/5.1/Data Mining/Assignment 2'

# Uncomment the following line to run in Google Colab.
# This will link the notebook to your Google drive to store your models and cache the dataset.
# This will ask you to authenticate and give permissions.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
base_dir = '/content/drive/My Drive/Colab Notebooks'
```

Mounted at /content/drive

## ▼ Using GPUs

While you can solve this assignment on a CPU, using a GPU will speed up training a lot. If you have a local GPU, you can use that. If you don't, we recommend Google Colab. When you are in Colab:

- In Runtime > Change runtime type, select the GPU under Hardware Accelerator
- Run the 4th cell on the top of this notebook to check that the GPU is found.

Note that the free version of Colab may not always have GPUs ready all the time, and may deny you a GPU when you have used them a lot.

When you are temporarily 'locked out', you can switch to a non-GPU runtime or to a local instance of Jupyter running on your machine. Take this into account in your planning, so that you don't do all your training at the last moment and are locked out right before the deadline.

```
# Uncomment the following to check whether you have access to a GPU in Google Colab
tf.config.experimental.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

## Grading

Grading is based on the following aspects:

- Correctness in answering the questions. Carefully read the questions and answer what is asked for. Train your models on the correct data. It should be clear on which data should be trained, but ask when in doubt. When something is not defined (e.g. the number of epochs or batch size), you can freely choose them.
- Clarity of your explanations. Write short but precise descriptions of what you did and why. Give short but clear explanations of the observed performance. After your explanation, your approach and model should make perfect sense.
- Part of your grade depends on how well your models perform. The top grade is awarded to the best model in class. You don't need to invest lots of effort into the last tiny improvement, though. Unless specified, we look at the accuracy on the validation set. If your learning curves are very erratic we'll compute a score based on the smoothed curves (i.e. single peaks don't count).
- The weight of each question is indicated. Take this into account when planning your time.

## Other tips

- Start now. Don't wait until the last minute to do the assignment. The models take time to train, most questions will require some thinking, and some require you to read up on some new concepts.
- Don't train for more than 100 epochs, i.e. don't throw excessing computational resources at the problem. If your model hasn't converged by then, think of ways it could be made to converge faster. In this assignment you are not after the last tiny improvement, you can stop when learning curves flatten out. Do at least 5 epochs to get a reasonable learning curve.

- Take care that you upload the results as requested. You need to submit not only the notebook but also the trained models.
- The dataset we will use is quite large and can take a while to download and cache for the first time, especially if you have limited RAM on your local device (laptop). Once cached, it should load fast.
- We provide an evaluation function that also stored models to disk. After you are done training the model, set the 'train' attribute to False so that the model doesn't train again (and loads from file instead) when you restart and rerun your notebook.
- Explore. For many questions we'll ask you to explain your model design decisions. You cannot magically know the best solutions but you can experiment based on your understanding and make decisions based on both your knowledge and experiments. Your explanation is at least as important as the performance of your model. Don't say 'I tried every possible option and this is the best I found', but rather 'I tried X because I think that Y' and then interpret the results and what you learned from them'.
- **Be original. We will check for plagiarism between student submissions.**

## ▼ Data

For this assignment we will use a few [Tensorflow Datasets](#). We selected the following datasets:

- [rock\\_paper\\_scissors](#) which includes images of hands playing rock, paper and scissor game. Each image is of shape (300, 300, 3) and the dataset contains 2520 training and 372 testing images. You can explore the images from this dataset at [this link](#).
- [tf\\_flowers](#) which includes images of 5 kinds of flowers (daisy, tulips, dandelions, roses, sunflowers) in a variety of settings. The images have different shapes, mostly around (260, 260, 3) and the dataset contains 3670 images. You can explore the images from this dataset at [this link](#).

```
import tensorflow_datasets as tfds

rock_train, rock_test = tfds.load(
    'rock_paper_scissors',
    split=['train', 'test'],
    shuffle_files=False,
)
flower_train, flower_test = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:]'],
    shuffle_files=False,
)
```

## ▼ Image preprocessing

We need to resize the images to fit the RAM memory provided by Google Colab. We reshape each dataset entry from (300, 300, 3) to (160, 160, 3). The `IMG_SIZE = 160` is chosen to be compatible with the trained weights of the model used for transfer learning at the end of the assignment. The value of each pixel is converted from [0, 255] range to [0, 1] range.

```
IMG_SIZE = 160
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

def process_img(sample):
    sample['image'] = tf.cast(sample['image'], tf.float32)
    sample['image'] = sample['image'] / 255.
    sample['image'] = tf.image.resize(sample['image'], [IMG_SIZE, IMG_SIZE])
    return sample

rock_train = rock_train.map(process_img)
rock_test = rock_test.map(process_img)
flower_train = flower_train.map(process_img)
flower_test = flower_test.map(process_img)
```

We randomly split the training dataset into 90% (X\_train, y\_train) and 10% (X\_valid, y\_valid). We also need to extract the labels from the data points since they are stored jointly. This yields training, validation, and test sets for both datasets.

```
# Don't change the name of these variables

from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Preprocesses and splits data into train, validation, and test splits
def split_data(train, test):

    # Converts TF objects to (X, y) data
    def numpy_convert(data, type):
        data_np = np.vstack(list(tfds.as_numpy(data)))
        return np.array(list(map(lambda x: x[0][type], data_np)))

    X_train_all = numpy_convert(train, 'image')
    y_train_all = to_categorical(numpy_convert(train, 'label'))
    X_train, X_valid, y_train, y_valid = train_test_split(X_train_all, y_train_all, stratify=y_train_all,
```

```

train_size=0.9, test_size=0.1, random_state=1)

X_test = numpy_convert(test,'image')
y_test = to_categorical(numpy_convert(test,'label'))
return X_train, X_valid, X_test, y_train, y_valid, y_test

Xr_train, Xr_valid, Xr_test, yr_train, yr_valid, yr_test = split_data(rock_train, rock_test)
Xf_train, Xf_valid, Xf_test, yf_train, yf_valid, yf_test = split_data(flower_train, flower_test)

class_names_r = ["rock", "paper", "scissors"]
class_names_f = ["dandelion", "daisy", "tulips", "sunflowers", "roses"]

```

Check the formatting - and what the data looks like

```

from random import randint

# Takes a list of row ids, and plots the corresponding images
# Use grayscale=True for plotting grayscale images
def plot_images(X, y, class_names, randomize= True, title = None):
    if randomize:
        images = [randint(0,len(X) - 1) for i in range(5)]
        X = [X[i] for i in images]
        y = [y[i] for i in images]
    fig, axes = plt.subplots(1, len(X),  figsize=(15,30))
    if title:
        plt.title(title)
    for n in range(len(X)):
        axes[n].imshow(X[n])
        axes[n].set_xlabel(class_names[np.argmax(y[n])])
        axes[n].set_xticks(()), axes[n].set_yticks(())
    plt.show()

plot_images(Xr_train, yr_train, class_names_r, title = "Training")
plot_images(Xr_valid, yr_valid, class_names_r, title = "Validation")
plot_images(Xr_test, yr_test, class_names_r, title = "Testing")

```



paper



scissors



scissors



rock



Training

rock



paper



rock



scissors

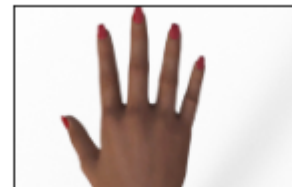


scissors



paper

Validation



Testing

```
print("Xr_train shape:", Xr_train.shape)
print("Xr_valid shape:", Xr_valid.shape)
print("Xr_test shape:", Xr_test.shape)
```

```
Xr_train shape: (2268, 160, 160, 3)
```

```
Xr_valid shape: (252, 160, 160, 3)
```

```
Xr_test shape: (372, 160, 160, 3)
```

```
plot_images(Xf_train, yf_train, class_names_f, title = "Training")
plot_images(Xf_valid, yf_valid, class_names_f, title = "Validation")
plot_images(Xf_test, yf_test, class_names_f, title = "Testing")
```



tulips



sunflowers



tulips

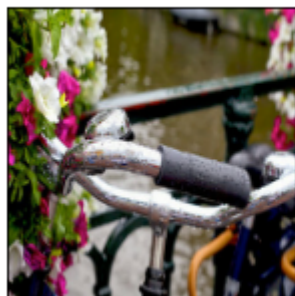


tulips

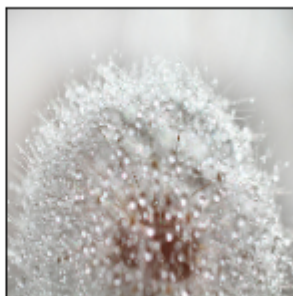


daisy

Training



tulips



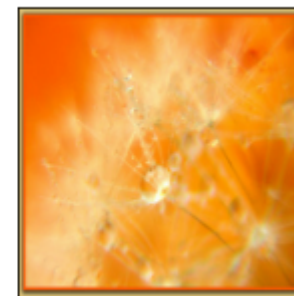
dandelion



dandelion

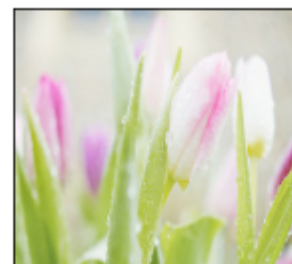


roses



dandelion

Validation



Testing

## ▼ Evaluation harness

We provide an evaluation function 'run\_evaluation' that you should use to evaluate all your models. It also stores the trained models to disk, to avoid having to train them over and over again. Your last run of the evaluation function (the last one stored to file), is the one that will be evaluated. The 'train' argument indicates whether to train or to load from disk. We have provided helper functions for saving and loading models to/from file, assuming you use TensorFlow. If you use PyTorch you'll have to adapt them.

```
# Set this to True to render and verify this notebook without retraining all the deep learning models.  
# All models will be loaded from file instead.  
stop_training = True
```



```

import os
import pickle
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model # for use with tensorflow
from tensorflow.keras.models import model_from_json
import pydot
from tensorflow.keras.utils import plot_model
from IPython.display import Image
import inspect
from IPython.core import page
page.page = print

# Helper function for user feedback
def shout(text, verbose=1):
    """ Prints text in red. Just for fun.
    """
    if verbose>0:
        print('\033[91m'+text+'\x1b[0m')

# Helper function to extract min/max from the learning curves
def minMax(x):
    return pd.Series(index=['min','max'],data=[x.min(),x.max()])

# Helper function to format answers
def print_answer(ans):
    output = ""
    for line in ans.splitlines()[0:]:
        output += line.strip() + " "
    output += "(length: "+str(len(output))+")\n"
    print(output)

def load_model_from_file(base_dir, name, extension='.h5'):
    """ Loads a model from a file. The returned model must have a 'fit' and 'summary'
    function following the Keras API. Don't change if you use TensorFlow. Otherwise,
    adapt as needed.
    Keyword arguments:
    base_dir -- Directory where the models are stored
    name -- Name of the model, e.g. 'question_1_1'

```

```

extension -- the file extension
"""
try:
    # if a json description is available, load config and then weights
    if os.path.isfile(os.path.join(base_dir, name+'.json')):
        json_file = open(os.path.join(base_dir, name+'.json'), 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        model = model_from_json(loaded_model_json)
        model.load_weights(os.path.join(base_dir, name+extension))
    # else just load the entire model from hdf5 file
    else:
        model = load_model(os.path.join(base_dir, name+extension))
except OSError:
    shout("Saved model could not be found. Was it trained and stored correctly? Is the base_dir correct?")
    return False
return model

```

```

def save_model_to_file(model, base_dir, name, extension='.h5'):
    """ Saves a model to file. Don't change if you use TensorFlow. Otherwise,
    adapt as needed.
    Keyword arguments:
    model -- the model to be saved
    base_dir -- Directory where the models should be stored
    name -- Name of the model, e.g. 'question_1_1'
    extension -- the file extension
    """
    path = os.path.join(base_dir, name+extension)
    model.save(path)
    size = os.path.getsize(path)
    # If model > 100MB, store the weights and architecture only.
    if size > 100*1024*1024:
        print("Model larger than 100MB, storing weights only.")
        model.save_weights(path)
        model_json = model.to_json()
        with open(os.path.join(base_dir, name+".json"), "w") as json_file:
            json_file.write(model_json)

```

# Evaluation harness

```

def run_evaluation(name, model_builder, data, base_dir, train=True,
                  generator=False, epochs=3, batch_size=32, steps_per_epoch=60,

```

```

verbose=1, print_model=True, **kwargs):
""" Trains and evaluates the given model on the predefined train and test splits,
stores the trained model and learning curves. Also prints out a summary of the
model and plots the learning curves.
Keyword arguments:
name -- the name of the model to be stored, e.g. 'question_1_1.h5'
model_builder -- function that returns an (untrained) model. The model must
                have a 'fit' function that follows the Keras API. It can wrap
                a non-Keras model as long as the 'fit' function takes the
                same attributes and returns the learning curves (history).
                It also must have a 'summary' function that prints out a
                model summary, and a 'save' function that saves the model
                to disk.
data -- data split for evaluation. A tuple of either:
        * Numpy arrays (X_train, X_val, y_train, y_val)
        * A data generator and validation data (generator, X_val, y_val)
base_dir -- the directory to save or read models to/from
train -- whether or not the data should be trained. If False, the trained model
        will be loaded from disk.
generator -- whether the data is given as a generator or not. Set batch size to None when using a generator.
epochs -- the number of epochs to train for
batch_size -- the batch size to train with. Set batch size to None when using a generator.
steps_per_epoch -- steps per epoch, in case a generator is used (ignored otherwise)
verbose -- verbosity level, 0: silent, 1: minimal,...
print_model -- whether or not to print the model
kwargs -- keyword arguments that should be passed to model_builder.
        Not required, but may help you to adjust its behavior
"""

model = model_builder(**kwargs)
if not model:
    shout("No model is returned by the model_builder")
    return
if not hasattr(model, 'fit'):
    shout("Model is not built correctly")
    return
learning_curves = {}

if train and not stop_training: # Train anew
    shout("Training the model", verbose)
    if generator:
        generator, X_val, y_val = data

```

```

        history = model.fit(generator, epochs=epochs, batch_size=batch_size,
                             steps_per_epoch=steps_per_epoch, verbose=1,
                             validation_data=(X_val, y_val))
        learning_curves = history.history
    else:
        X_train, X_val, y_train, y_val = data
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
                             verbose=1, validation_data=(X_val, y_val))
        learning_curves = history.history
    shout("Saving to file", verbose)
    save_model_to_file(model, base_dir, name)
    with open(os.path.join(base_dir, name+'.p'), 'wb') as file_pi:
        pickle.dump(learning_curves, file_pi)
    shout("Model stored in "+base_dir, verbose)
else: # Load from file
    model = load_model_from_file(base_dir, name)
    if not model:
        shout("Model not found")
        return
    learning_curves = None
    try:
        learning_curves = pickle.load(open(os.path.join(base_dir, name+'.p'), "rb"))
    except FileNotFoundError:
        shout("Learning curves not found")
        return

# Report
lc = pd.DataFrame(learning_curves)
print("Max val score: {:.2f}%".format(lc.iloc[:,3].max()*100))
lc.plot(lw=2,style=['b:', 'r:', 'b-', 'r-']);
plt.xlabel('epochs');
plt.show()

if print_model:
    print(model.summary())
plot_model(model, to_file=os.path.join(base_dir, name+'.png'))

```

## ▼ Part 1. Convolutional neural networks (30 points)

### Question 1.1: Design a ConvNet (15 points)

- Build a sequential convolutional neural network to distinguish the classes of hand gestures and flowers. It's best to do this by implementing a function that builds and returns the model (see the example above).
- You can build two entirely different models for both datasets.
- Try to achieve the best validation accuracy you can. You can use any depth, any combination of layers, and any kind of regularization and tuning. You can use different batch sizes and number of epochs. Think carefully about all design decisions.
- Beware that you don't overfit on the validation set. You can occasionally test on the test set to see if you get a similar score, but don't tune on the test set.
- In your report provide a description of your final model (you can include screenshots) and clearly explain all your design choices: explain what you did and also why. Also discuss the performance of the model. Is it working well? Did other models work much worse? Both the performance of the model and your explanations matter.
- Explicitly explore different filter sizes and padding techniques. Explain what the effect is, also in relationship with the layer input sizes.
- Explain the differences between both models (for hand gestures and flowers). How did the dataset influence your decisions? What works better on one dataset but not on the other?
- The report and code (in this notebook) will count for 10 points, and model performance for 5 points.
- We will look at the correctness and cleanliness of the code. If we cannot understand the code, this may hurt your grade.

**NOTE:** The training might be noisy and unstable. The training dataset is quite small, with a lot of variety in the data (e.g. different positions of the hand, skin color, left and right hand,...). For this exercise, we will not use any data augmentation, so avoiding overfitting is quite hard. However, you are expected to tackle the overfitting by layer regularization, dropout layers, learning rate tuning and more.

```
import random

random.seed(10)
np.random.seed(10)
tf.random.set_seed(10)
```

## ▼ Final CNN Rock

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
```

```
def build_convnetmodel_hand():
    model = models.Sequential()
    model.add(layers.Conv2D(4, (3, 3), activation='relu', input_shape=IMG_SHAPE))
    model.add(layers.Dropout(0.2))
    model.add(layers.Conv2D(8, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(3, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
eval_split_r = Xr_train, Xr_valid, yr_train, yr_valid
```

```
run_evaluation("model_1_rock", build_convnetmodel_hand, eval_split_r, base_dir, train=True, epochs=5, batch_size=32)
```

### Training the model

Epoch 1/5

71/71 [=====] - 2s 27ms/step - loss: 1.8692 - accuracy: 0.5529 - val\_loss: 0.5289 - val\_accuracy: 0.8929

Epoch 2/5

71/71 [=====] - 2s 22ms/step - loss: 0.2007 - accuracy: 0.9590 - val\_loss: 0.0795 - val\_accuracy: 0.9881

Epoch 3/5

71/71 [=====] - 2s 23ms/step - loss: 0.0334 - accuracy: 0.9947 - val\_loss: 0.0368 - val\_accuracy: 0.9921

Epoch 4/5

71/71 [=====] - 2s 22ms/step - loss: 0.0130 - accuracy: 0.9991 - val\_loss: 0.0324 - val\_accuracy: 0.9960

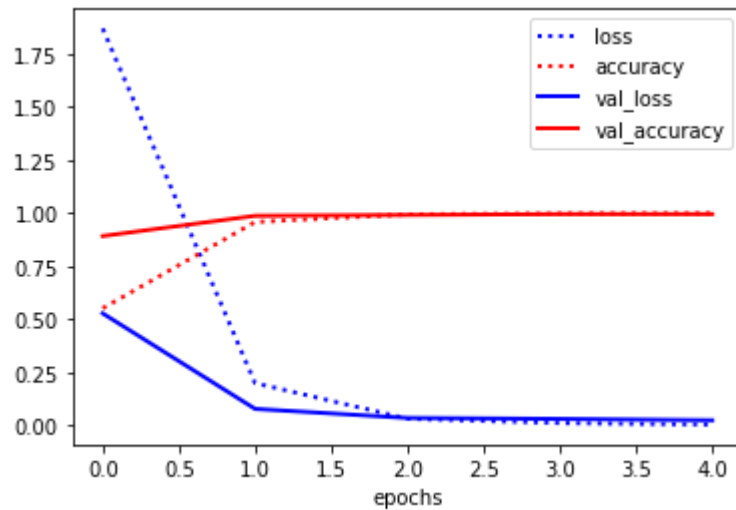
Epoch 5/5

71/71 [=====] - 2s 23ms/step - loss: 0.0054 - accuracy: 1.0000 - val\_loss: 0.0259 - val\_accuracy: 0.9960

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 99.60%



Model: "sequential\_14"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

## Final CNN Flower

```
def build_convnetmodel_flower():  
    model = models.Sequential()  
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=IMG_SHAPE))  
    model.add(layers.MaxPooling2D((2, 2), strides=2))  
    model.add(layers.Dropout(0.1))
```

```
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=2))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=2))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=2))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu', kernel_regularizer = tf.keras.regularizers.L2(0.01)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(265, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

return model
```

```
eval_split_f = Xf_train, Xf_valid, yf_train, yf_valid
```

```
run_evaluation("model_1_flower", build_convnetmodel_flower, eval_split_f, base_dir, train=True, epochs=25, batch_size=32)
```



## Training the model

Epoch 1/25

83/83 [=====] - 4s 41ms/step - loss: 3.1392 - accuracy: 0.3361 - val\_loss: 1.6118 - val\_accuracy: 0.3776

Epoch 2/25

83/83 [=====] - 3s 38ms/step - loss: 1.3779 - accuracy: 0.4720 - val\_loss: 1.3396 - val\_accuracy: 0.5136

Epoch 3/25

83/83 [=====] - 3s 38ms/step - loss: 1.2564 - accuracy: 0.5091 - val\_loss: 1.2647 - val\_accuracy: 0.5340

Epoch 4/25

83/83 [=====] - 3s 38ms/step - loss: 1.1547 - accuracy: 0.5776 - val\_loss: 1.1890 - val\_accuracy: 0.5918

Epoch 5/25

83/83 [=====] - 3s 38ms/step - loss: 1.1017 - accuracy: 0.6079 - val\_loss: 1.0528 - val\_accuracy: 0.6293

Epoch 6/25

83/83 [=====] - 3s 38ms/step - loss: 1.0479 - accuracy: 0.6571 - val\_loss: 1.0065 - val\_accuracy: 0.7109

Epoch 7/25

83/83 [=====] - 3s 38ms/step - loss: 1.0333 - accuracy: 0.6597 - val\_loss: 1.0319 - val\_accuracy: 0.6905

Epoch 8/25

83/83 [=====] - 3s 38ms/step - loss: 0.9899 - accuracy: 0.6794 - val\_loss: 0.9676 - val\_accuracy: 0.6769

Epoch 9/25

83/83 [=====] - 3s 38ms/step - loss: 0.9437 - accuracy: 0.7017 - val\_loss: 0.9334 - val\_accuracy: 0.6837

Epoch 10/25

83/83 [=====] - 3s 39ms/step - loss: 0.9271 - accuracy: 0.7097 - val\_loss: 0.9441 - val\_accuracy: 0.7075

Epoch 11/25

83/83 [=====] - 3s 39ms/step - loss: 0.9321 - accuracy: 0.7203 - val\_loss: 1.0116 - val\_accuracy: 0.6769

Epoch 12/25

83/83 [=====] - 3s 39ms/step - loss: 0.9192 - accuracy: 0.7282 - val\_loss: 0.9345 - val\_accuracy: 0.6973

Epoch 13/25

83/83 [=====] - 3s 39ms/step - loss: 0.8673 - accuracy: 0.7593 - val\_loss: 0.9051 - val\_accuracy: 0.7449

Epoch 14/25

83/83 [=====] - 3s 38ms/step - loss: 0.8760 - accuracy: 0.7472 - val\_loss: 0.9348 - val\_accuracy: 0.7347

Epoch 15/25

83/83 [=====] - 3s 39ms/step - loss: 0.8743 - accuracy: 0.7653 - val\_loss: 0.9082 - val\_accuracy: 0.7449

Epoch 16/25

83/83 [=====] - 3s 39ms/step - loss: 0.8563 - accuracy: 0.7687 - val\_loss: 0.9171 - val\_accuracy: 0.7347

Epoch 17/25

83/83 [=====] - 3s 38ms/step - loss: 0.8193 - accuracy: 0.7888 - val\_loss: 0.8413 - val\_accuracy: 0.7721

Epoch 18/25

83/83 [=====] - 3s 39ms/step - loss: 0.7768 - accuracy: 0.8028 - val\_loss: 0.9161 - val\_accuracy: 0.7551

Epoch 19/25

83/83 [=====] - 3s 39ms/step - loss: 0.8246 - accuracy: 0.7922 - val\_loss: 0.9046 - val\_accuracy: 0.7755

Epoch 20/25

83/83 [=====] - 3s 39ms/step - loss: 0.7779 - accuracy: 0.8191 - val\_loss: 0.8960 - val\_accuracy: 0.7551

Epoch 21/25

83/83 [=====] - 3s 41ms/step - loss: 0.8108 - accuracy: 0.8070 - val\_loss: 0.9441 - val\_accuracy: 0.7619

Epoch 22/25

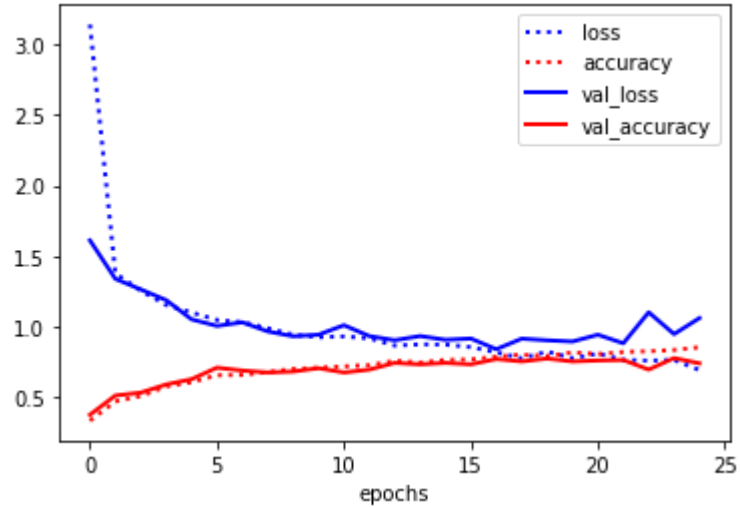
83/83 [=====] - 3s 38ms/step - loss: 0.7659 - accuracy: 0.8217 - val\_loss: 0.8835 - val\_accuracy: 0.7653

Epoch 23/25  
83/83 [=====] - 3s 38ms/step - loss: 0.7609 - accuracy: 0.8270 - val\_loss: 1.1031 - val\_accuracy: 0.6973  
Epoch 24/25  
83/83 [=====] - 3s 38ms/step - loss: 0.7657 - accuracy: 0.8361 - val\_loss: 0.9484 - val\_accuracy: 0.7789  
Epoch 25/25  
83/83 [=====] - 3s 38ms/step - loss: 0.6943 - accuracy: 0.8550 - val\_loss: 1.0620 - val\_accuracy: 0.7415

Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 77.89%



Model: "sequential\_15"

Layer (type)	Output Shape	Param #
=====		
conv2d_14 (Conv2D)	(None, 160, 160, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 80, 80, 32)	0
dropout_16 (Dropout)	(None, 80, 80, 32)	0
conv2d_15 (Conv2D)	(None, 80, 80, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 40, 40, 64)	0
dropout_17 (Dropout)	(None, 40, 40, 64)	0
conv2d_16 (Conv2D)	(None, 40, 40, 128)	73856
max_pooling2d_10 (MaxPooling2D)	(None, 20, 20, 128)	0

dropout_18 (Dropout)	(None, 20, 20, 128)	0
conv2d_17 (Conv2D)	(None, 20, 20, 128)	147584
max_pooling2d_11 (MaxPooling2D)	(None, 10, 10, 128)	0
dropout_19 (Dropout)	(None, 10, 10, 128)	0
flatten_15 (Flatten)	(None, 12800)	0
dense_28 (Dense)	(None, 512)	6554112

## Question 1.2: Data Augmentation (15 points)

- Augment the preprocessed training data. You can explore using image shifts, rotations, zooming, flips, color augmentation etc. What works well, and what does not? Reason about what might work or not on these images and also verify it through experimentation.
- Implement the generator in a function `augment_data` that returns a generator and the validation set
- Evaluate the model with the augmented data using the 'run\_evaluation' function (see the example below).
- In your report, add a clear explanation of your design choices for augmentation techniques. Also discuss the performance of the model. Did augmentation help?
- Did you find that different augmentations help for the two different datasets?
- The report and code will count for 10 points, and model performance for 5 points. Again, correctness and cleanliness of the code matter.

## Final Rock Aug

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def augment_data(X_train, y_train, X_valid, y_valid):
    """ Augments the data and returns a generator and the validation data and labels
    """

    # Boilerplate code. You can change this however you like.
    generator = ImageDataGenerator(zoom_range=0.2).flow(X_train, y_train)
    return generator, X_valid, y_valid
```

```
def augment_data_rock():  
    return augment_data(Xr_train, yr_train, Xr_valid, yr_valid)  
  
# Batch size should be None since we are using a generator.  
# You can add a steps_per_epoch parameter if you like  
run_evaluation("model_2_rock", build_convnetmodel_hand, augment_data_rock(), base_dir,  
               train=True, generator=True, epochs=5, batch_size=None, steps_per_epoch=71)
```

### Training the model

Epoch 1/5

71/71 [=====] - 13s 169ms/step - loss: 7.9242 - accuracy: 0.4480 - val\_loss: 1.0987 - val\_accuracy: 0.3770

Epoch 2/5

71/71 [=====] - 12s 162ms/step - loss: 0.8666 - accuracy: 0.6283 - val\_loss: 1.0175 - val\_accuracy: 0.4683

Epoch 3/5

71/71 [=====] - 12s 164ms/step - loss: 0.7175 - accuracy: 0.7037 - val\_loss: 0.7997 - val\_accuracy: 0.5635

Epoch 4/5

71/71 [=====] - 12s 164ms/step - loss: 0.6003 - accuracy: 0.7500 - val\_loss: 0.7948 - val\_accuracy: 0.5437

Epoch 5/5

71/71 [=====] - 12s 162ms/step - loss: 0.5116 - accuracy: 0.7932 - val\_loss: 0.5596 - val\_accuracy: 0.7302

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 73.02%



## Final Flower Aug

```
def augment_data_f(X_train, y_train, X_valid, y_valid):  
    """ Augments the data and returns a generator and the validation data and labels  
    """  
    # Boilerplate code. You can change this however you like.  
    generator = ImageDataGenerator(shear_range=0.2, horizontal_flip=True).flow(X_train, y_train)  
    return generator, X_valid, y_valid  
  
def augment_data_flower():  
    return augment_data_f(Xf_train, yf_train, Xf_valid, yf_valid)  
  
# Batch size should be None since we are using a generator.  
# You can add a steps_per_epoch parameter if you like  
steps = 83  
run_evaluation("model_2_flower", build_convnetmodel_flower, augment_data_flower(), base_dir, train=True, generator=True, epochs=25, batch_size=
```

## Training the model

Epoch 1/25

83/83 [=====] - 15s 169ms/step - loss: 3.0377 - accuracy: 0.2911 - val\_loss: 1.4985 - val\_accuracy: 0.3810

Epoch 2/25

83/83 [=====] - 14s 167ms/step - loss: 1.4361 - accuracy: 0.4580 - val\_loss: 1.4376 - val\_accuracy: 0.4456

Epoch 3/25

83/83 [=====] - 14s 165ms/step - loss: 1.2853 - accuracy: 0.4996 - val\_loss: 1.2191 - val\_accuracy: 0.5068

Epoch 4/25

83/83 [=====] - 14s 166ms/step - loss: 1.2186 - accuracy: 0.5522 - val\_loss: 1.2445 - val\_accuracy: 0.5476

Epoch 5/25

83/83 [=====] - 14s 166ms/step - loss: 1.1360 - accuracy: 0.5829 - val\_loss: 1.0867 - val\_accuracy: 0.5850

Epoch 6/25

83/83 [=====] - 14s 165ms/step - loss: 1.0937 - accuracy: 0.6276 - val\_loss: 1.0953 - val\_accuracy: 0.6088

Epoch 7/25

83/83 [=====] - 14s 168ms/step - loss: 1.0456 - accuracy: 0.6397 - val\_loss: 1.0851 - val\_accuracy: 0.6701

Epoch 8/25

83/83 [=====] - 14s 166ms/step - loss: 1.0427 - accuracy: 0.6518 - val\_loss: 1.0073 - val\_accuracy: 0.6497

Epoch 9/25

83/83 [=====] - 14s 168ms/step - loss: 0.9832 - accuracy: 0.6718 - val\_loss: 0.9344 - val\_accuracy: 0.7041

Epoch 10/25

83/83 [=====] - 16s 197ms/step - loss: 0.9661 - accuracy: 0.6934 - val\_loss: 0.9187 - val\_accuracy: 0.7279

Epoch 11/25

83/83 [=====] - 14s 168ms/step - loss: 0.9669 - accuracy: 0.6756 - val\_loss: 0.9508 - val\_accuracy: 0.6973

Epoch 12/25

83/83 [=====] - 14s 166ms/step - loss: 0.9384 - accuracy: 0.6927 - val\_loss: 0.9282 - val\_accuracy: 0.7347

Epoch 13/25

83/83 [=====] - 14s 166ms/step - loss: 0.8975 - accuracy: 0.7135 - val\_loss: 0.8686 - val\_accuracy: 0.7143

Epoch 14/25

83/83 [=====] - 14s 167ms/step - loss: 0.8936 - accuracy: 0.7180 - val\_loss: 0.9042 - val\_accuracy: 0.7381

Epoch 15/25

83/83 [=====] - 14s 167ms/step - loss: 0.9189 - accuracy: 0.7040 - val\_loss: 0.9331 - val\_accuracy: 0.7041

Epoch 16/25

83/83 [=====] - 14s 167ms/step - loss: 0.8999 - accuracy: 0.7252 - val\_loss: 0.8612 - val\_accuracy: 0.7279

Epoch 17/25

83/83 [=====] - 14s 166ms/step - loss: 0.8775 - accuracy: 0.7305 - val\_loss: 0.9194 - val\_accuracy: 0.7177

Epoch 18/25

83/83 [=====] - 14s 168ms/step - loss: 0.8609 - accuracy: 0.7362 - val\_loss: 0.8530 - val\_accuracy: 0.7551

Epoch 19/25

83/83 [=====] - 14s 166ms/step - loss: 0.8319 - accuracy: 0.7411 - val\_loss: 0.8485 - val\_accuracy: 0.7551

Epoch 20/25

83/83 [=====] - 14s 168ms/step - loss: 0.8499 - accuracy: 0.7593 - val\_loss: 0.8322 - val\_accuracy: 0.7517

Epoch 21/25

83/83 [=====] - 14s 165ms/step - loss: 0.8499 - accuracy: 0.7513 - val\_loss: 0.8651 - val\_accuracy: 0.7619

Epoch 22/25

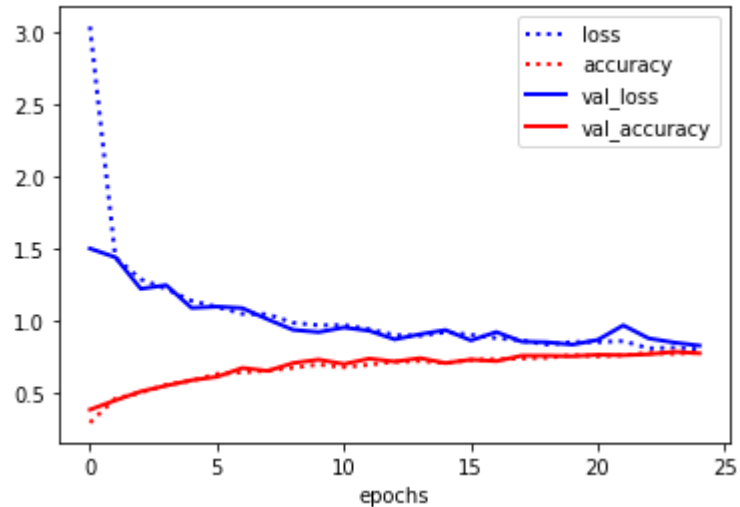
83/83 [=====] - 14s 167ms/step - loss: 0.8565 - accuracy: 0.7581 - val\_loss: 0.9658 - val\_accuracy: 0.7585

Epoch 23/25  
83/83 [=====] - 14s 166ms/step - loss: 0.8062 - accuracy: 0.7748 - val\_loss: 0.8755 - val\_accuracy: 0.7653  
Epoch 24/25  
83/83 [=====] - 14s 166ms/step - loss: 0.8107 - accuracy: 0.7680 - val\_loss: 0.8465 - val\_accuracy: 0.7823  
Epoch 25/25  
83/83 [=====] - 14s 166ms/step - loss: 0.8003 - accuracy: 0.7767 - val\_loss: 0.8253 - val\_accuracy: 0.7721

Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 78.23%



Model: "sequential\_17"

Layer (type)	Output Shape	Param #
=====		
conv2d_20 (Conv2D)	(None, 160, 160, 32)	896
max_pooling2d_12 (MaxPoolin g2D)	(None, 80, 80, 32)	0
dropout_23 (Dropout)	(None, 80, 80, 32)	0
conv2d_21 (Conv2D)	(None, 80, 80, 64)	18496
max_pooling2d_13 (MaxPoolin g2D)	(None, 40, 40, 64)	0
dropout_24 (Dropout)	(None, 40, 40, 64)	0
conv2d_22 (Conv2D)	(None, 40, 40, 128)	73856
max_pooling2d_14 (MaxPoolin g2D)	(None, 20, 20, 128)	0

dropout_25 (Dropout)	(None, 20, 20, 128)	0
conv2d_23 (Conv2D)	(None, 20, 20, 128)	147584
max_pooling2d_15 (MaxPoolin g2D)	(None, 10, 10, 128)	0
dropout_26 (Dropout)	(None, 10, 10, 128)	0
flatten_17 (Flatten)	(None, 12800)	0
dense_33 (Dense)	(None, 512)	6554112
dropout_27 (Dropout)	(None, 512)	0
dense_34 (Dense)	(None, 265)	135945
dense_35 (Dense)	(None, 5)	1330

```
=====
Total params: 6,932,219
Trainable params: 6,932,219
Non-trainable params: 0
None
```

## ▼ Part 2. Model evaluation (25 points)

### Question 2.1: Accuracy on the testing set (5 points)

Load the last trained model (e.g. `model_2_rock` in the example above) and use it to predict the classes for the test set. Compute the accuracy based on the test set.

- Discuss the observed performance. Is it what you expected? Is your model under- or overfitting?
- 2 points will be given for your discussion, 3 for the test set performance.

### Question 2.2: Analyze errors (20 points)

- Plot the confusion matrix and discuss which classes are often confused.



- Analyze the misclassifications in more depth by visualizing which kinds of mistakes are made for each class. Plot 5 examples of misclassifications for each class and interpret them. For instance, are the errors related to the background, noisiness, etc.? Repeat for both datasets.
- Interpret the results and summarize your findings in your report. Focus on explaining why certain images are misclassified. Are they somehow harder or has your model not learned to recognize some occurrences in the images? Keep in mind that there can be images which are quite similar in the dataset. Do you notice different kinds of misclassifications between the two datasets (e.g. what is the effect of not having a white background)?

**Note:** If, for some classes, you cannot find 5 misclassifications, simply plot all misclassifications for that class.

## ▼ For Rock dataset

```
from statistics import mean
from sklearn.metrics import accuracy_score

model_2_rock = load_model(os.path.join(base_dir, "model_1_rock.h5"))

pred_r = model_2_rock.predict(Xr_test)

rock = [prediction[0] for prediction in pred_r]
paper = [prediction[1] for prediction in pred_r]
scissor = [prediction[2] for prediction in pred_r]

model_2_rock.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
eval = model_2_rock.evaluate(Xr_test, yr_test)
print(eval)
f"rock: {mean(rock)}, paper: {mean(paper)}, scissor: {mean(scissor)}"

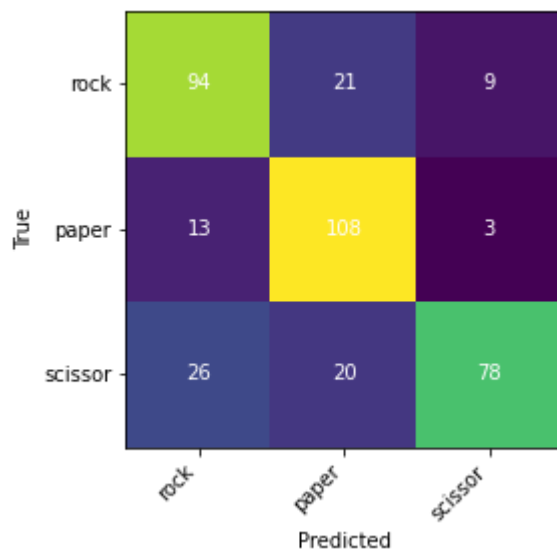
12/12 [=====] - 0s 7ms/step
12/12 [=====] - 0s 9ms/step - loss: 0.8434 - accuracy: 0.7527
[0.8433789610862732, 0.7526881694793701]
'rock: 0.3904173672199249, paper: 0.35767313838005066, scissor: 0.251909464597702 '
```

```

from sklearn.metrics import confusion_matrix
hand_classes = {0: "rock", 1: "paper", 2: "scissor"}

cm = confusion_matrix(np.argmax(yr_test, axis=1), np.argmax(pred_r, axis=1))
fig, ax = plt.subplots()
im = ax.imshow(cm)
ax.set_xticks(np.arange(3)), ax.set_yticks(np.arange(3))
ax.set_xticklabels(list(hand_classes.values()), rotation=45, ha="right")
ax.set_yticklabels(list(hand_classes.values()))
ax.set_ylabel('True')
ax.set_xlabel('Predicted')
for i in range(9):
    ax.text(int(i/3), i%3, cm[i%3, int(i/3)], ha="center", va="center", color="w")

```



```

def misclassplot(X_test, y_test, pred, classes, classlist):
    fig, axes = plt.subplots(1, 5, figsize=(10, 5))
    misclassified_samples = np.nonzero(np.argmax(y_test, axis=1) != np.argmax(pred, axis=1))[0]

    all_class = [idx for idx, x in enumerate(y_test) if (np.argmax(y_test, axis=1)[idx] == classes)]

    misclassified_class = [x for idx, x in enumerate(all_class) if (x in list(misclassified_samples))]

    if misclassified_class:
        for nr, i in enumerate(misclassified_class[:5]):

```

```

axes[nr].imshow(X_test[i])
axes[nr].set_xlabel("Predicted: %s,\n Actual : %s" % (classlist[np.argmax(pred[i])],classlist[np.argmax(y_test[i])]))
axes[nr].set_xticks(()), axes[nr].set_yticks(())

else:
    print("No misclassifications")

plt.show();

misclassplot(Xr_test, yr_test, pred_r, 0, hand_classes)
misclassplot(Xr_test, yr_test, pred_r, 1, hand_classes)
misclassplot(Xr_test, yr_test, pred_r, 2, hand_classes)

```



▼ For the Flower dataset

```

model_2_flower = load_model(os.path.join(base_dir, "model_2_flower.h5"))

```

```

pred_f = model_2_flower.predict(Xf_test)

dandelion = [prediction[0] for prediction in pred_f]
daisy = [prediction[1] for prediction in pred_f]
tulips = [prediction[2] for prediction in pred_f]
sunflowers = [prediction[3] for prediction in pred_f]
roses = [prediction[4] for prediction in pred_f]

model_2_flower.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
eval = model_2_flower.evaluate(Xf_test, yf_test)
print(eval)
f"dandelion: {mean(dandelion)}, daisy: {mean(daisy)}, tulips: {mean(tulips)}, sunflowers: {mean(sunflowers)}, roses: {mean(roses)}"

23/23 [=====] - 0s 14ms/step
23/23 [=====] - 1s 16ms/step - loss: 0.8638 - accuracy: 0.7534
[0.8637890815734863, 0.7534059882164001]
'dandelion: 0.3007013499736786, daisy: 0.15020598471164703, tulips: 0.204207181930542, sunflowers: 0.17717935144901276, roses: 0.1677061
1703395844'

flower_classes = {0: "dandelion", 1: "daisy", 2: "tulips", 3: "sunflowers", 4: "roses"}
cm = confusion_matrix(np.argmax(yf_test, axis=1), np.argmax(pred_f, axis=1))
fig, ax = plt.subplots()
im = ax.imshow(cm)
ax.set_xticks(np.arange(5)), ax.set_yticks(np.arange(5))
ax.set_xticklabels(list(flower_classes.values()), rotation=45, ha="right")
ax.set_yticklabels(list(flower_classes.values()))
ax.set_ylabel('True')
ax.set_xlabel('Predicted')
for i in range(25):
    ax.text(int(i/5), i%5, cm[i%5, int(i/5)], ha="center", va="center", color="w")

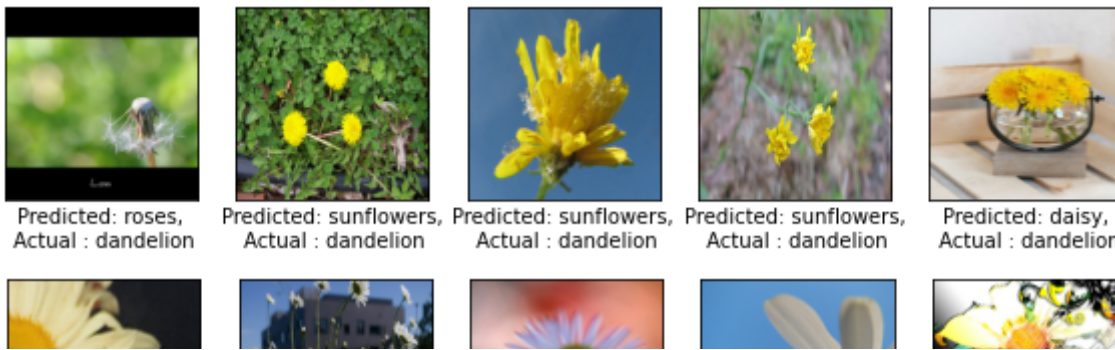
```

True \ Predict	dandelion	142	6	1	9	1
	daisy	21	92	1	1	5
	tulips	28	1	113	5	39

```

misclassplot(Xf_test, yf_test, pred_f, 0, flower_classes)
misclassplot(Xf_test, yf_test, pred_f ,1, flower_classes)
misclassplot(Xf_test, yf_test, pred_f, 2, flower_classes)
misclassplot(Xf_test, yf_test, pred_f, 3, flower_classes)
misclassplot(Xf_test, yf_test, pred_f, 4, flower_classes)

```



## Part 3. Transfer learning (45 points)

### Question 3.1 Transfer learning from MobileNet (25 points)

- Import the MobileNetV2 model, pretrained on ImageNet. [See here](#). Only import the convolutional part, not the dense layers.
- Build a model that adds at least one dense hidden layer and output layer to the convolutional base, and freezes the convolutional base. Add [Global Average Pooling](#) after the convolutional base, right before the dense layer(s).
- Consider unfreezing the last few convolutional layers, in a systematic way, and evaluate whether that works better. You can also consider adding multiple dense hidden layers and regularization layers.
- Train the resulting model on the augmented training data and report the model and score.
- Explain all your design decisions clearly in your report. Do you need a different approach for the two datasets (e.g. different amounts of finetuning)?
- Explore [other pretrained models](#). Can you get better performance? Do keep in mind that many will be too large to run in Google Colab.
- The report and code will count for 20 points, and model performance for 5 points.



## Final Rock Pretrained model MobileNetV2



```
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

conv_base = MobileNetV2(weights = 'imagenet', include_top = False, input_shape = (160, 160, 3))

def build_convmodel_rockV2():
    model = models.Sequential()
    model.add(conv_base)
```

```
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(224, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
conv_base.trainable = False
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
return model
```

```
# run_evaluation("model_3_rock", build_convmodel_rockV2, augment_data_rock(), base_dir, train=True, generator=True, epochs=3, batch_size=32, st
run_evaluation("model_3_rock_MNV2", build_convmodel_rockV2, eval_split_r, base_dir, train=True, epochs=8, batch_size=32)
```

### Training the model

Epoch 1/8

71/71 [=====] - 11s 43ms/step - loss: 0.1508 - accuracy: 0.9511 - val\_loss: 0.0096 - val\_accuracy: 1.0000

Epoch 2/8

71/71 [=====] - 2s 30ms/step - loss: 0.0045 - accuracy: 0.9991 - val\_loss: 0.0048 - val\_accuracy: 1.0000

Epoch 3/8

71/71 [=====] - 2s 30ms/step - loss: 0.0019 - accuracy: 1.0000 - val\_loss: 0.0028 - val\_accuracy: 1.0000

Epoch 4/8

71/71 [=====] - 2s 30ms/step - loss: 0.0011 - accuracy: 1.0000 - val\_loss: 0.0021 - val\_accuracy: 1.0000

Epoch 5/8

71/71 [=====] - 2s 30ms/step - loss: 7.5112e-04 - accuracy: 1.0000 - val\_loss: 0.0012 - val\_accuracy: 1.0000

Epoch 6/8

71/71 [=====] - 2s 30ms/step - loss: 5.2453e-04 - accuracy: 1.0000 - val\_loss: 9.0074e-04 - val\_accuracy: 1.0000

Epoch 7/8

71/71 [=====] - 2s 30ms/step - loss: 3.8644e-04 - accuracy: 1.0000 - val\_loss: 7.2679e-04 - val\_accuracy: 1.0000

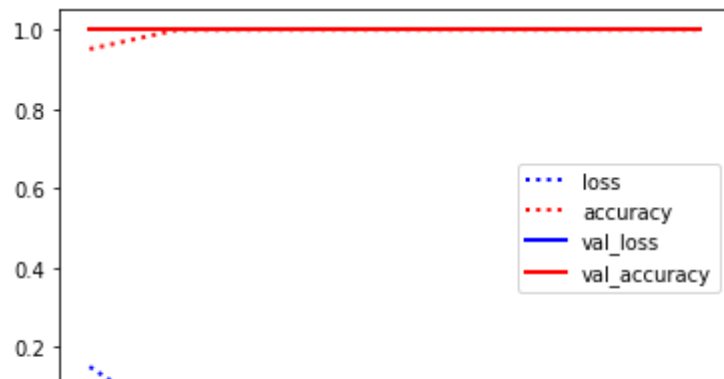
Epoch 8/8

71/71 [=====] - 2s 30ms/step - loss: 3.0751e-04 - accuracy: 1.0000 - val\_loss: 5.7087e-04 - val\_accuracy: 1.0000

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 100.00%



## Final Flower Pretrained model MobileNetV2

Model: "sequential\_2"

```
def build_convmodel_flowerV2():
    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Flatten())
    model.add(layers.Dense(224, activation='relu', kernel_regularizer = tf.keras.regularizers.L2(0.01)))
    model.add(layers.Dropout(0.4))
    model.add(layers.Dense(5, activation='softmax'))
```



```
conv_base.trainable = False
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
return model
```

```
run_evaluation("model_3_flower_MNV2", build_convmodel_flowerV2, augment_data_flower(), base_dir, train=True, generator=True, epochs=8, batch_si
```

### Training the model

Epoch 1/8

83/83 [=====] - 18s 173ms/step - loss: 3.4179 - accuracy: 0.7566 - val\_loss: 2.2718 - val\_accuracy: 0.8639

Epoch 2/8

83/83 [=====] - 14s 163ms/step - loss: 1.7880 - accuracy: 0.8645 - val\_loss: 1.4298 - val\_accuracy: 0.8673

Epoch 3/8

83/83 [=====] - 14s 171ms/step - loss: 1.1420 - accuracy: 0.8974 - val\_loss: 1.0398 - val\_accuracy: 0.8741

Epoch 4/8

83/83 [=====] - 14s 164ms/step - loss: 0.8467 - accuracy: 0.9016 - val\_loss: 0.8455 - val\_accuracy: 0.8503

Epoch 5/8

83/83 [=====] - 14s 162ms/step - loss: 0.7248 - accuracy: 0.8940 - val\_loss: 0.7344 - val\_accuracy: 0.8844

Epoch 6/8

83/83 [=====] - 14s 163ms/step - loss: 0.6022 - accuracy: 0.9167 - val\_loss: 0.6920 - val\_accuracy: 0.8639

Epoch 7/8

83/83 [=====] - 14s 169ms/step - loss: 0.5355 - accuracy: 0.9194 - val\_loss: 0.6624 - val\_accuracy: 0.8571

Epoch 8/8

83/83 [=====] - 14s 163ms/step - loss: 0.5285 - accuracy: 0.9107 - val\_loss: 0.6222 - val\_accuracy: 0.8810

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 88.44%



## Rock Pretrained models

```
from tensorflow.keras.applications.efficientnet import EfficientNetB2
```

```
conv_base = EfficientNetB2(weights = 'imagenet', include_top = False, input_shape = (160, 160, 3))
```

```
def build_convmodel_rockV2():  
    model = models.Sequential()  
    model.add(conv_base)  
    model.add(layers.GlobalAveragePooling2D())  
    model.add(layers.Flatten())  
    model.add(layers.Dense(224, activation='relu'))  
    model.add(layers.Dense(3, activation='softmax'))  
    conv_base.trainable = False
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
return model
```

```
# run_evaluation("model_3_rock_ENB2", build_convmodel_rockV2, augment_data_rock(), base_dir, train=True, generator=True, epochs=8, batch_size=32)  
run_evaluation("model_3_rock_ENB2", build_convmodel_rockV2, eval_split_r, base_dir, train=True, epochs=8, batch_size=32)
```

### Training the model

Epoch 1/8

7/7 [=====] - 13s 614ms/step - loss: 1.2578 - accuracy: 0.3438 - val\_loss: 1.1754 - val\_accuracy: 0.3333

Epoch 2/8

7/7 [=====] - 2s 284ms/step - loss: 1.1311 - accuracy: 0.3036 - val\_loss: 1.1103 - val\_accuracy: 0.3333

Epoch 3/8

7/7 [=====] - 2s 330ms/step - loss: 1.1094 - accuracy: 0.3839 - val\_loss: 1.1094 - val\_accuracy: 0.3333

Epoch 4/8

7/7 [=====] - 3s 381ms/step - loss: 1.0761 - accuracy: 0.4152 - val\_loss: 1.1287 - val\_accuracy: 0.3770

Epoch 5/8

7/7 [=====] - 2s 256ms/step - loss: 1.1305 - accuracy: 0.3259 - val\_loss: 1.0968 - val\_accuracy: 0.3333

Epoch 6/8

7/7 [=====] - 2s 277ms/step - loss: 1.1208 - accuracy: 0.3438 - val\_loss: 1.0923 - val\_accuracy: 0.3690

Epoch 7/8

7/7 [=====] - 2s 276ms/step - loss: 1.0951 - accuracy: 0.3864 - val\_loss: 1.0848 - val\_accuracy: 0.4444

Epoch 8/8

7/7 [=====] - 2s 279ms/step - loss: 1.0884 - accuracy: 0.4273 - val\_loss: 1.1320 - val\_accuracy: 0.3333

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 44.44%

```
from tensorflow.keras.applications.densenet import DenseNet121
```

```
conv_base = DenseNet121(weights = 'imagenet', include_top = False, input_shape = (160, 160, 3))
```

```
def build_convmodel_rockV2():
```

```
    model = models.Sequential()
```

```
    model.add(conv_base)
```

```
    model.add(layers.GlobalAveragePooling2D())
```

```
    model.add(layers.Flatten())
```

```
    model.add(layers.Dense(224, activation='relu'))
```

```
    model.add(layers.Dense(3, activation='softmax'))
```

```
    conv_base.trainable = False
```

```
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
    return model
```

```
# run_evaluation("model_3_rock_DN121", build_convmodel_rockV2, augment_data_rock(), base_dir, train=True, generator=True, epochs=8, batch_size=32)
```

```
run_evaluation("model_3_rock_DN121", build_convmodel_rockV2, eval_split_r, base_dir, train=True, epochs=8, batch_size=32)
```

### Training the model

Epoch 1/8

71/71 [=====] - 22s 212ms/step - loss: 0.1969 - accuracy: 0.9286 - val\_loss: 0.0406 - val\_accuracy: 0.9881

Epoch 2/8

71/71 [=====] - 13s 185ms/step - loss: 0.0264 - accuracy: 0.9965 - val\_loss: 0.0124 - val\_accuracy: 1.0000

Epoch 3/8

71/71 [=====] - 13s 185ms/step - loss: 0.0109 - accuracy: 0.9987 - val\_loss: 0.0039 - val\_accuracy: 1.0000

Epoch 4/8

71/71 [=====] - 13s 188ms/step - loss: 0.0064 - accuracy: 1.0000 - val\_loss: 0.0028 - val\_accuracy: 1.0000

Epoch 5/8

71/71 [=====] - 13s 182ms/step - loss: 0.0048 - accuracy: 1.0000 - val\_loss: 0.0014 - val\_accuracy: 1.0000

Epoch 6/8

71/71 [=====] - 13s 185ms/step - loss: 0.0027 - accuracy: 1.0000 - val\_loss: 9.1847e-04 - val\_accuracy: 1.0000

Epoch 7/8

71/71 [=====] - 13s 182ms/step - loss: 0.0016 - accuracy: 1.0000 - val\_loss: 9.8755e-04 - val\_accuracy: 1.0000

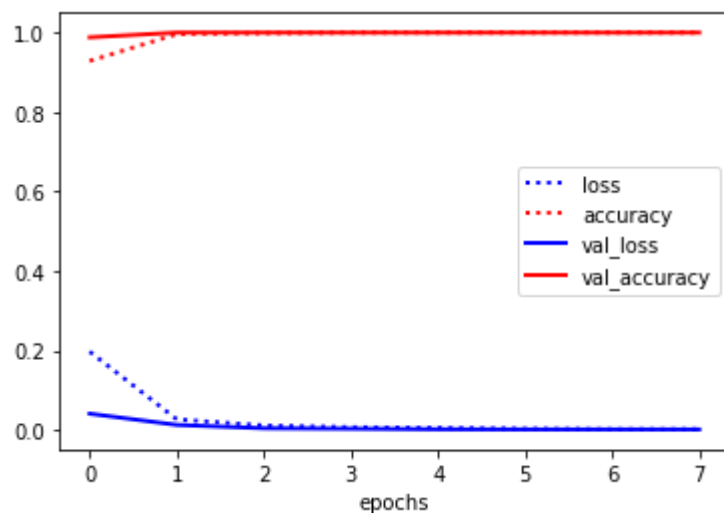
Epoch 8/8

71/71 [=====] - 13s 180ms/step - loss: 0.0017 - accuracy: 1.0000 - val\_loss: 9.3097e-04 - val\_accuracy: 1.0000

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 100.00%



Model: "sequential\_21"

Layer (type)	Output Shape	Param #
densenet121 (Functional)	(None, 5, 5, 1024)	7037504
global_average_pooling2d_13 (GlobalAveragePooling2D)	(None, 1024)	0
flatten_21 (Flatten)	(None, 1024)	0

dense_42 (Dense)	(None, 224)	229600
------------------	-------------	--------

dense_43 (Dense)	(None, 3)	675
------------------	-----------	-----

=====  
Total params: 7,267,779  
Trainable params: 230,275  
Non-trainable params: 7,037,504

## ▼ Flower Pretrained models

```
from tensorflow.keras.applications.efficientnet import EfficientNetB2
```

```
conv_base = EfficientNetB2(weights = 'imagenet', include_top = False, input_shape = (160, 160, 3))
```

```
def build_convmodel_flowerV2():  
    model = models.Sequential()  
    model.add(conv_base)  
    model.add(layers.GlobalAveragePooling2D())  
    model.add(layers.Flatten())  
    model.add(layers.Dense(224, activation='relu', kernel_regularizer = tf.keras.regularizers.L2(0.01)))  
    model.add(layers.Dropout(0.4))  
    model.add(layers.Dense(5, activation='softmax'))  
    conv_base.trainable = False  
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
    return model
```

```
run_evaluation("model_3_flower_ENB2", build_convmodel_flowerV2, augment_data_flower(), base_dir, train=True, generator=True, epochs=8, batch_si
```

Downloading data from [https://storage.googleapis.com/keras-applications/efficientnetb2\\_notop.h5](https://storage.googleapis.com/keras-applications/efficientnetb2_notop.h5)

31790344/31790344 [=====] - 0s 0us/step

### Training the model

Epoch 1/8

83/83 [=====] - 29s 212ms/step - loss: 2.9155 - accuracy: 0.2335 - val\_loss: 1.8891 - val\_accuracy: 0.1905

Epoch 2/8

83/83 [=====] - 15s 175ms/step - loss: 1.7506 - accuracy: 0.2324 - val\_loss: 1.6654 - val\_accuracy: 0.2517

Epoch 3/8

83/83 [=====] - 15s 175ms/step - loss: 1.6437 - accuracy: 0.2332 - val\_loss: 1.6200 - val\_accuracy: 0.2517

Epoch 4/8

83/83 [=====] - 15s 175ms/step - loss: 1.6186 - accuracy: 0.2385 - val\_loss: 1.6104 - val\_accuracy: 0.2517

Epoch 5/8

83/83 [=====] - 15s 175ms/step - loss: 1.6118 - accuracy: 0.2498 - val\_loss: 1.6079 - val\_accuracy: 0.2517

Epoch 6/8

83/83 [=====] - 15s 178ms/step - loss: 1.6059 - accuracy: 0.2517 - val\_loss: 1.6020 - val\_accuracy: 0.2517

Epoch 7/8

83/83 [=====] - 16s 192ms/step - loss: 1.6027 - accuracy: 0.2532 - val\_loss: 1.6037 - val\_accuracy: 0.2517

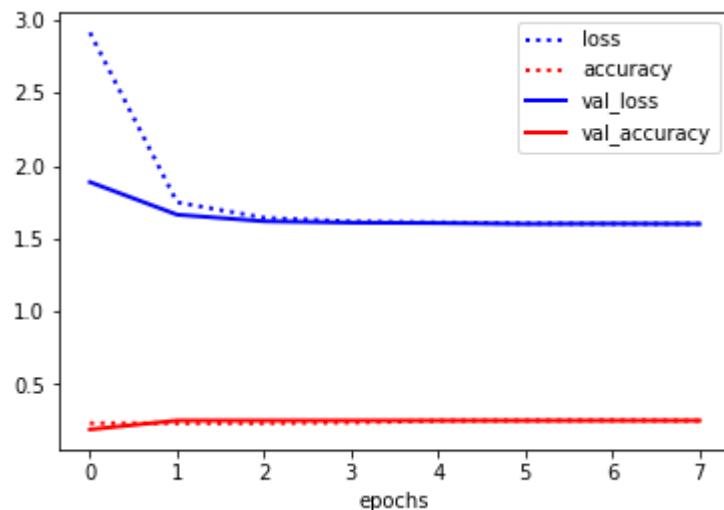
Epoch 8/8

83/83 [=====] - 15s 176ms/step - loss: 1.6033 - accuracy: 0.2502 - val\_loss: 1.6019 - val\_accuracy: 0.2517

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 25.17%



Model: "sequential\_5"

Layer (type)	Output Shape	Param #
efficientnetb2 (Functional)	(None, 5, 5, 1408)	7768569
global_average_pooling2d_5 (GlobalAveragePooling2D)	(None, 1408)	0

flatten_5 (Flatten)	(None, 1408)	0
dense_10 (Dense)	(None, 224)	315616
dropout_1 (Dropout)	(None, 224)	0

```
from tensorflow.keras.applications.densenet import DenseNet121
```

```
conv_base = DenseNet121(weights = 'imagenet', include_top = False, input_shape = (160, 160, 3))
```

```
def build_convmodel_flowerV2():  
    model = models.Sequential()  
    model.add(conv_base)  
    model.add(layers.GlobalAveragePooling2D())  
    model.add(layers.Flatten())  
    model.add(layers.Dense(224, activation='relu', kernel_regularizer = tf.keras.regularizers.L2(0.01)))  
    model.add(layers.Dropout(0.4))  
    model.add(layers.Dense(5, activation='softmax'))  
    conv_base.trainable = False  
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
    return model
```

```
run_evaluation("model_3_flower_DN121", build_convmodel_flowerV2, augment_data_flower(), base_dir, train=True, generator=True, epochs=8, batch_s
```



Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernel](https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernel)  
29084464/29084464 [=====] - 0s 0us/step

### Training the model

Epoch 1/8

83/83 [=====] - 26s 219ms/step - loss: 2.8074 - accuracy: 0.7010 - val\_loss: 1.6057 - val\_accuracy: 0.8741

Epoch 2/8

83/83 [=====] - 15s 175ms/step - loss: 1.4596 - accuracy: 0.8486 - val\_loss: 1.1941 - val\_accuracy: 0.8741

Epoch 3/8

83/83 [=====] - 15s 176ms/step - loss: 1.0933 - accuracy: 0.8668 - val\_loss: 0.9577 - val\_accuracy: 0.8810

Epoch 4/8

83/83 [=====] - 15s 174ms/step - loss: 0.8667 - accuracy: 0.8827 - val\_loss: 0.8209 - val\_accuracy: 0.8707

Epoch 5/8

83/83 [=====] - 15s 175ms/step - loss: 0.7245 - accuracy: 0.8948 - val\_loss: 0.6624 - val\_accuracy: 0.9048

Epoch 6/8

83/83 [=====] - 16s 189ms/step - loss: 0.6178 - accuracy: 0.9005 - val\_loss: 0.6163 - val\_accuracy: 0.8878

Epoch 7/8

83/83 [=====] - 15s 178ms/step - loss: 0.5540 - accuracy: 0.9073 - val\_loss: 0.5816 - val\_accuracy: 0.8844

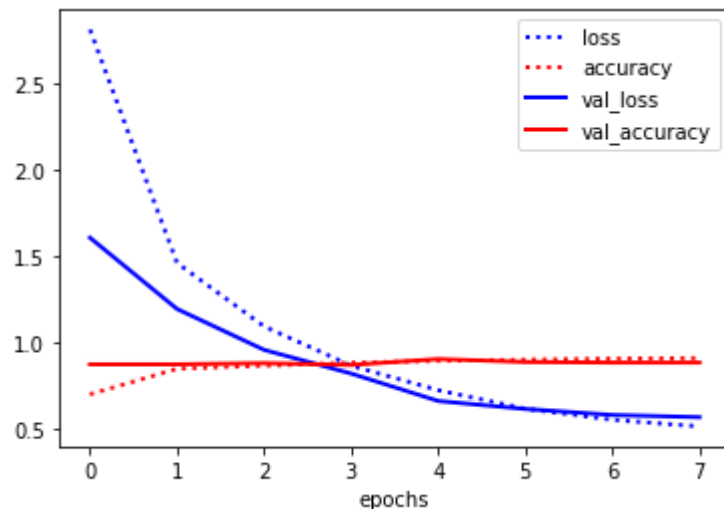
Epoch 8/8

83/83 [=====] - 15s 176ms/step - loss: 0.5154 - accuracy: 0.9103 - val\_loss: 0.5691 - val\_accuracy: 0.8844

### Saving to file

Model stored in /content/drive/My Drive/Colab Notebooks

Max val score: 90.48%



Model: "sequential\_6"

Layer (type)	Output Shape	Param #
=====	=====	=====
densenet121 (Functional)	(None, 5, 5, 1024)	7037504
global average pooling2d	(None, 1024)	0

## ▼ Question 3.2: Visualizing the learned embeddings with tSNE (20 points)

Extract the learned embeddings of the training images using your pretrained model and plot them on a 2D map using [tSNE](#) as the dimensionality reduction technique.

- Extract the embeddings based on the (finetuned) convolutional part of your models (e.g. MobileNetV2 generates 1280-sized embeddings). The embeddings are the output of the GlobalAveragePooling layer.
- Applies scikit-learn's implementation of [tSNE](#) to reduce the size of the embeddings from 1280 to 2 (e.g for MobileNetV2 this will mean `original_array` of size (num\_images, 1280) compressed to a reduced array of size (num\_images, 2)).
- Scatterplot the 2D vectors on a map highlighting the formed clusters, and color-coded by the true labels.
- Interpret the 2D TSNE map in terms of the formed clusters. Discuss the performance of the transfer learning model in your report. Does the learned embedding clearly separate the different classes, and can you also see this in the 2D-plot (e.g. do they cluster together)?
- Include the plot and your clear explanations in your report.

Dubbeltklik (of druk op Enter) om te bewerken

```
from sklearn.manifold import TSNE
from tensorflow import keras
import seaborn as sns

def tsneplot(model_name, global_layer, X_train, y_train, shape):
    model = load_model_from_file(base_dir, model_name, extension = '.h5')
    extract_layer = keras.Model(inputs=model.input, outputs=model.get_layer(global_layer).output)
    embedding_output = extract_layer.predict(X_train)

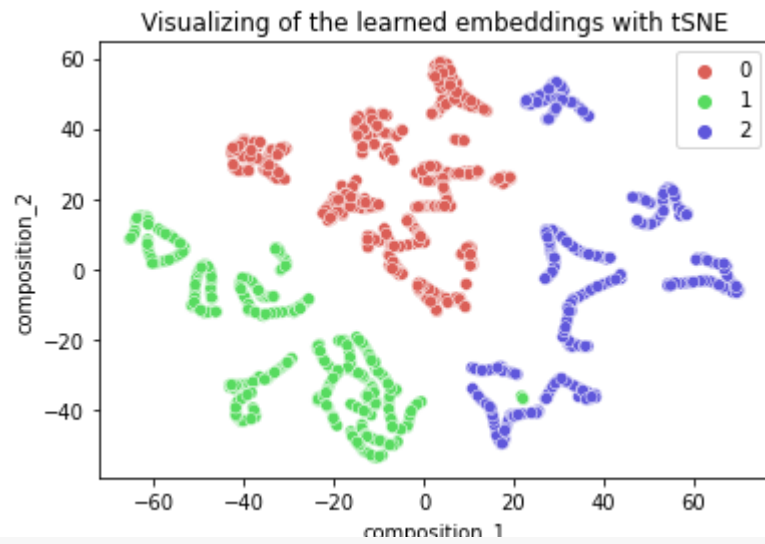
    tsne = TSNE(n_components = 2, learning_rate = 'auto', init='random')
    tsne_embedded = tsne.fit_transform(embedding_output)

    df = pd.DataFrame()
    df["y"] = [i for item in y_train for i in range(shape) if item[i] == 1]
    df["composition_1"] = tsne_embedded[:,0]
    df["composition_2"] = tsne_embedded[:,1]

    sns.scatterplot(x = "composition_1", y = "composition_2", hue=df.y.tolist(), palette=sns.color_palette("hls", shape), data = df).set(title =

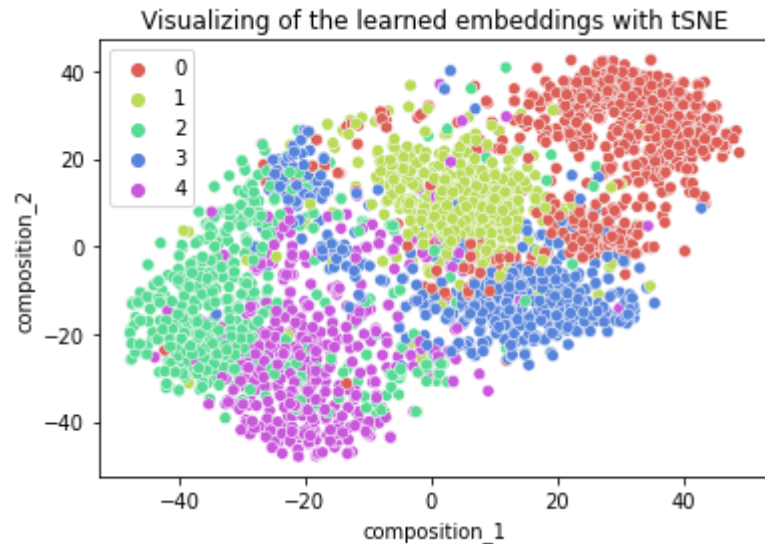
tsneplot('model_3_rock_MNV2', 'global_average_pooling2d', Xr_train, yr_train, 3)
```

71/71 [=====] - 3s 22ms/step



```
tsneplot('model_3_flower_MNV2', 'global_average_pooling2d_2', Xf_train, yf_train, 5)
```

83/83 [=====] - 2s 18ms/step



### ▼ Bonus question (+5 points)

Use some of the model interpretation techniques we saw in class, e.g. plotting the filter activations or drawing class activation maps. Interpret the results. Are your models indeed learning what they are supposed to learn?



```

        plt.title("Activation of layer {} ({}).format(layer_index+1,layer_name), fontdict={'fontsize':30*fig_scale})
        plt.grid(False)
        plt.axis("off")
        plt.imshow(display_grid, aspect='auto', cmap='viridis')

plt.show()

def plot_activations(model_name):
    model = load_model(os.path.join(base_dir, model_name))
    img_path = os.path.join(base_dir, '27465811_9477c9d044.jpg')

    img = image.load_img(img_path, target_size=(160, 160))

    img_tensor = image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.
    layer_outputs = [layer.output for layer in model.layers[:8]]
    activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
    activations = activation_model.predict(img_tensor)

    layer_names = []

    for layer in model.layers[:8]:
        layer_names.append(layer.name)
    for i in [0,3, 6]:
        plot_layers(i, activations, layer_names)

```

```

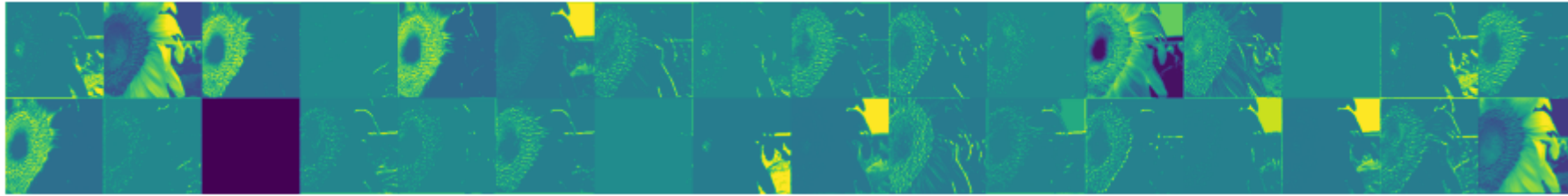
plot_activations("model_1_flower.h5")

```

1/1 [=====] - 0s 81ms/step

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:31: RuntimeWarning: invalid value encountered in true\_divide

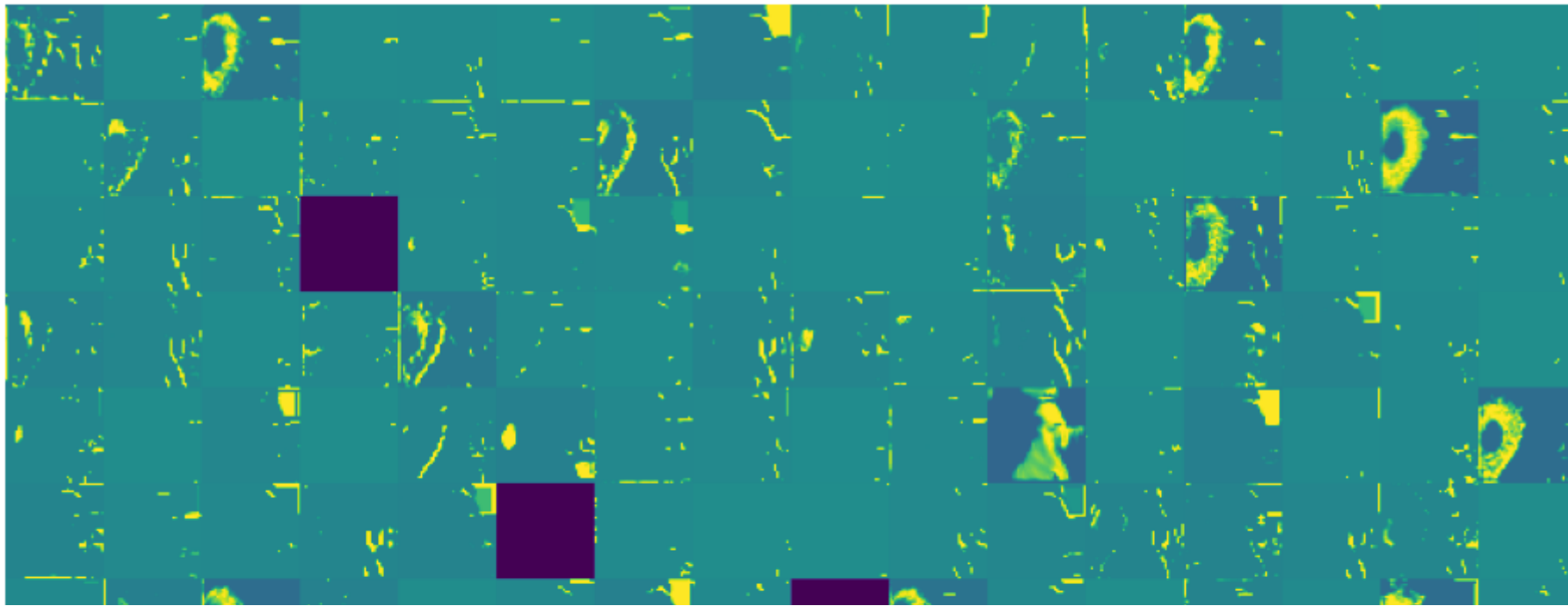
Activation of layer 1 (conv2d\_14)



Activation of layer 4 (conv2d\_15)



Activation of layer 7 (conv2d\_16)



## ▼ Explanation

As shown in the images above, the first layer focusses on broader details like the outline of the flower. In the 4th layer the focus is on the pistil of the flower and the leaves, we see that the model is making decisions in this layer mostly based on the granularity of the pistil. In the 7th layer we see that it's focussing on the finest details of the image. Think of the flower edges. This is also obvious when looking at the different classes. As a sunflower is the only flower in the class range with a round big pistil. That is why the model is focussing for sunflowers on the pistil and not other parts of the flower.

Have fun!