In [ ]:

```
# Fill in your names using the format below
your_name_1 = "Bastiaansen, Thijs"
your_name_2 = "Jacobs, Iris Catharina Johan"
your_name_3 = "van Pinxteren, Milo"
your_name_4 = "Rijkers, Niek"
your_name_5 = "Thijssen, Aisja"
```

# 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.

In [ ]:

```
# 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    print("Tensorflow is outdated.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.
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
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'`

In [ ]:

```
# For storing the files in a local folder.
# base_dir = './'

# 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 data
# 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/assignment'
```

```
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.

In [ ]:

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

Out[ ]:

```
[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 assigment 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.

In [ ]:

```
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,
)
```

```
Downloading and preparing dataset 219.53 MiB (download: 219.53 MiB, generated: Unknown siz
e, total: 219.53 MiB) to ~/tensorflow_datasets/rock_paper_scissors/3.0.0...
```

```
Dataset rock_paper_scissors downloaded and prepared to ~/tensorflow_datasets/rock_paper_sc
issors/3.0.0. Subsequent calls will reuse this data.
```

```
Downloading and preparing dataset 218.21 MiB (download: 218.21 MiB, generated: 221.83 MiB,
total: 440.05 MiB) to ~/tensorflow_datasets/tf_flowers/3.0.1...

Dataset tf_flowers downloaded and prepared to ~/tensorflow_datasets/tf_flowers/3.0.1. Subs
equent calls will reuse this data.
```

### 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 assigment. The value of each pixel is converted from [0, 255] range to [0, 1] range.

In [ ]:

```
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.

In [ ]:

```
# 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
                                                        train_size=0.9, test_size=0.1, ran

  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_tes
Xf_train, Xf_valid, Xf_test, yf_train, yf_valid, yf_test = split_data(flower_train, flowe

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

Check the formatting - and what the data looks like

In [ ]:

```
from random import randint
```

```python
# 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")
```



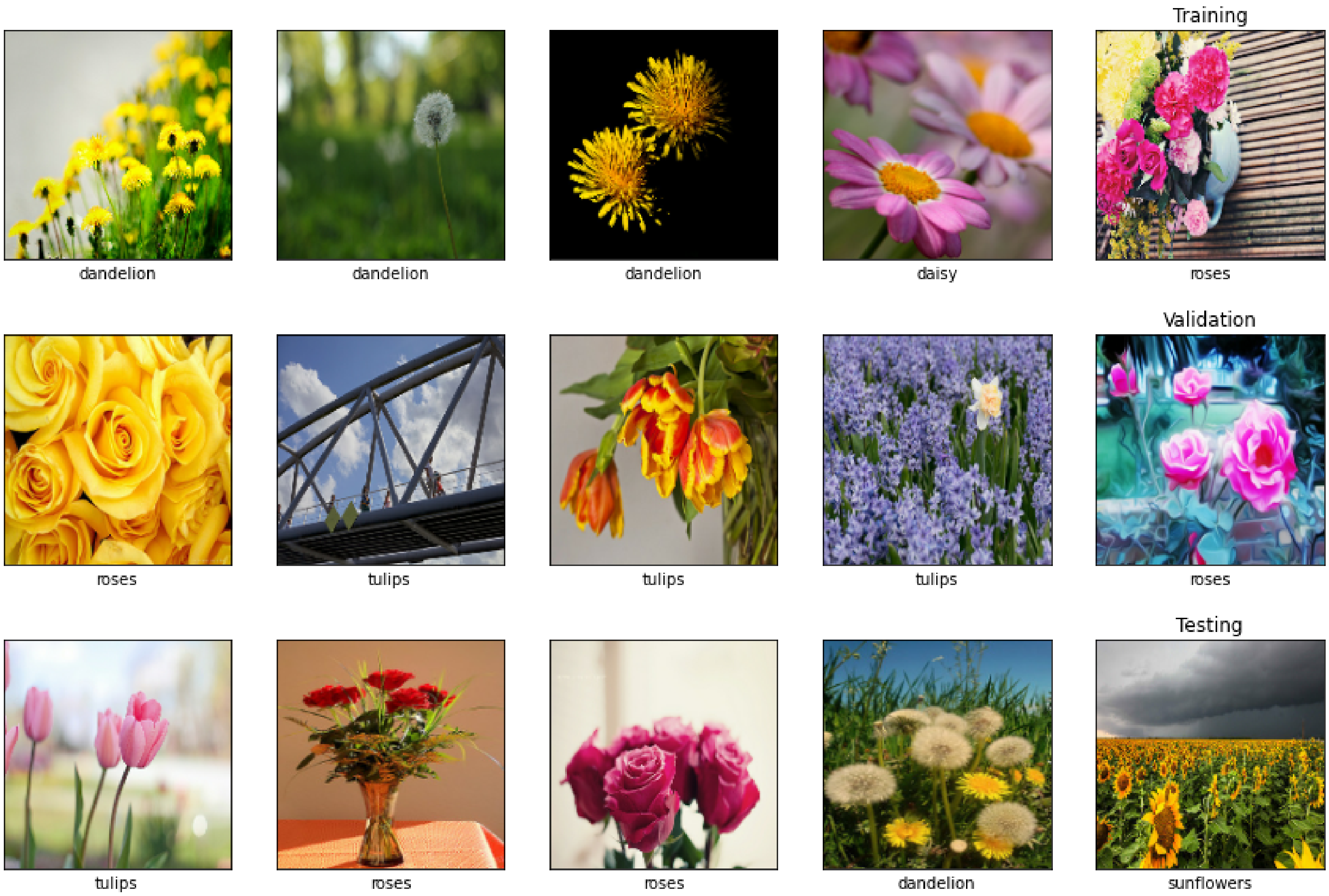

In [ ]:

```python
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)
```

In [ ]:

```python
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")
```




## 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.

In [ ]:

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

In [ ]:

```
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:
```

```python
        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
        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 w
    epochs -- the number of epochs to train for
    batch_size -- the batch size to train with. Set batch size to None when using a genera
    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 expolain 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.
- Explictly 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.

In [ ]:

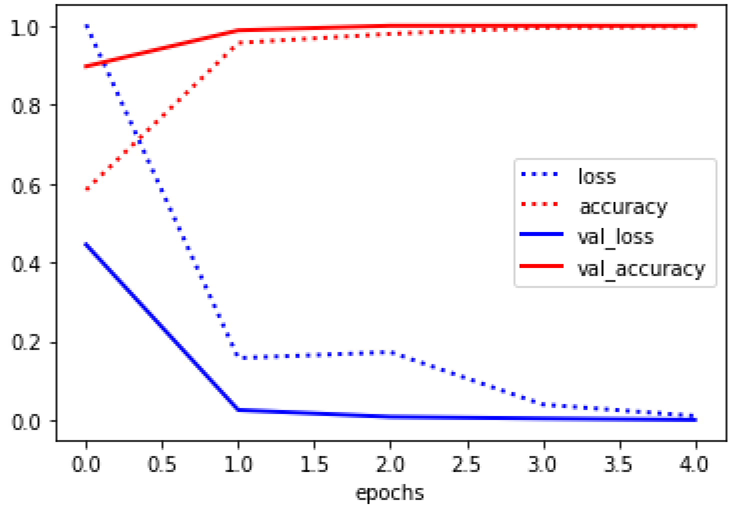
```
from tensorflow.keras import models
from tensorflow.keras import layers

# The run_evaluation method expects a model, so we implement a function that builds and re
# This also makes it easier to work with multiple models or to change parameters

# Function for rock, paper, scissors model
def build_r_model():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE,IMG_SIZE,
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.3))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
```

```python
    model.add(layers.Dropout(0.4))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(3, activation='softmax'))
    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# function for flower model
def build_f_model():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE,IMG_SIZE,
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.3))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.4))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.4))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(5, activation='softmax'))
    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Packing the training and validation sets as required by the run_evaluation method
eval_split_r = Xr_train, Xr_valid, yr_train, yr_valid
eval_split_f = Xf_train, Xf_valid, yf_train, yf_valid
```

In [ ]:

```python
# Build and store rock paper scissors model
run_evaluation("model_final5_rock", build_r_model, eval_split_r, base_dir,
               train=True, epochs=5, batch_size=32)
```

```
Training the model
Epoch 1/5
71/71 [==============================] - 11s 41ms/step - loss: 1.0030 - accuracy: 0.5833 -
 val_loss: 0.4448 - val_accuracy: 0.8968
Epoch 2/5
71/71 [==============================] - 2s 33ms/step - loss: 0.1569 - accuracy: 0.9563 -
 val_loss: 0.0250 - val_accuracy: 0.9881
Epoch 3/5
71/71 [==============================] - 2s 32ms/step - loss: 0.1718 - accuracy: 0.9793 -
 val_loss: 0.0076 - val_accuracy: 1.0000
Epoch 4/5
71/71 [==============================] - 2s 32ms/step - loss: 0.0391 - accuracy: 0.9951 -
 val_loss: 0.0034 - val_accuracy: 1.0000
Epoch 5/5
71/71 [==============================] - 2s 33ms/step - loss: 0.0100 - accuracy: 0.9969 -
 val_loss: 9.4760e-05 - val_accuracy: 1.0000
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 100.00%
```



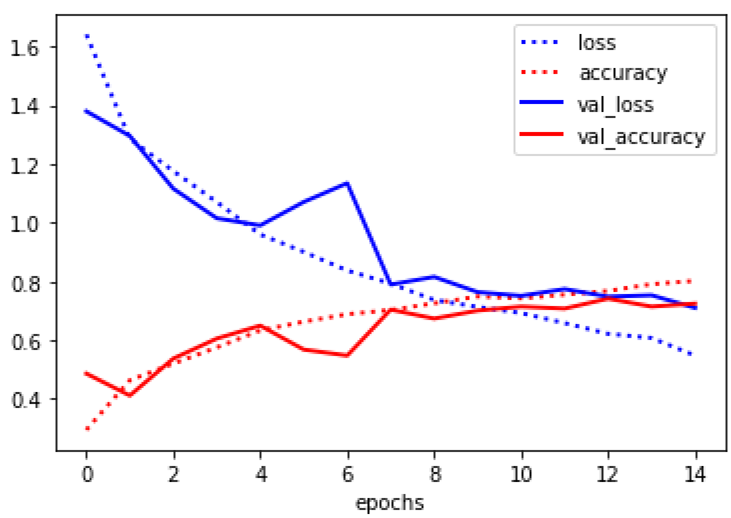

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 158, 158, 32)      896

 max_pooling2d (MaxPooling2D  (None, 79, 79, 32)       0
 )

 dropout (Dropout)           (None, 79, 79, 32)        0

 conv2d_1 (Conv2D)           (None, 77, 77, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 38, 38, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 38, 38, 64)        0

 conv2d_2 (Conv2D)           (None, 36, 36, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 18, 18, 64)        0

 flatten (Flatten)           (None, 20736)             0

 dense (Dense)               (None, 64)                1327168

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 1,383,683
Trainable params: 1,383,683
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
# Build and store flower model
run_evaluation("model_final5_flower", build_f_model, eval_split_f, base_dir,
               train=True, epochs=15, batch_size=32)
```

```
Training the model
Epoch 1/15
83/83 [==============================] - 5s 44ms/step - loss: 1.6405 - accuracy: 0.2949 -
 val_loss: 1.3792 - val_accuracy: 0.4864
```

```
Epoch 2/15
83/83 [==============================] - 3s 36ms/step - loss: 1.2861 - accuracy: 0.4637 -
 val_loss: 1.2953 - val_accuracy: 0.4116
Epoch 3/15
83/83 [==============================] - 3s 36ms/step - loss: 1.1755 - accuracy: 0.5204 -
 val_loss: 1.1159 - val_accuracy: 0.5374
Epoch 4/15
83/83 [==============================] - 3s 35ms/step - loss: 1.0703 - accuracy: 0.5753 -
 val_loss: 1.0151 - val_accuracy: 0.6054
Epoch 5/15
83/83 [==============================] - 3s 36ms/step - loss: 0.9597 - accuracy: 0.6332 -
 val_loss: 0.9906 - val_accuracy: 0.6497
Epoch 6/15
83/83 [==============================] - 3s 36ms/step - loss: 0.9007 - accuracy: 0.6628 -
 val_loss: 1.0704 - val_accuracy: 0.5680
Epoch 7/15
83/83 [==============================] - 3s 36ms/step - loss: 0.8371 - accuracy: 0.6885 -
 val_loss: 1.1344 - val_accuracy: 0.5476
Epoch 8/15
83/83 [==============================] - 3s 38ms/step - loss: 0.7927 - accuracy: 0.7025 -
 val_loss: 0.7893 - val_accuracy: 0.7041
Epoch 9/15
83/83 [==============================] - 3s 36ms/step - loss: 0.7369 - accuracy: 0.7256 -
 val_loss: 0.8155 - val_accuracy: 0.6735
Epoch 10/15
83/83 [==============================] - 3s 36ms/step - loss: 0.7132 - accuracy: 0.7487 -
 val_loss: 0.7635 - val_accuracy: 0.7007
Epoch 11/15
83/83 [==============================] - 4s 42ms/step - loss: 0.6923 - accuracy: 0.7422 -
 val_loss: 0.7509 - val_accuracy: 0.7143
Epoch 12/15
83/83 [==============================] - 4s 43ms/step - loss: 0.6583 - accuracy: 0.7551 -
 val_loss: 0.7740 - val_accuracy: 0.7075
Epoch 13/15
83/83 [==============================] - 3s 39ms/step - loss: 0.6218 - accuracy: 0.7676 -
 val_loss: 0.7480 - val_accuracy: 0.7415
Epoch 14/15
83/83 [==============================] - 3s 41ms/step - loss: 0.6076 - accuracy: 0.7907 -
 val_loss: 0.7530 - val_accuracy: 0.7143
Epoch 15/15
83/83 [==============================] - 3s 39ms/step - loss: 0.5478 - accuracy: 0.8020 -
 val_loss: 0.7099 - val_accuracy: 0.7245
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 74.15%
```




```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d_3 (Conv2D)           (None, 158, 158, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 79, 79, 32)       0
 2D)

 dropout_4 (Dropout)         (None, 79, 79, 32)        0

 conv2d_4 (Conv2D)           (None, 77, 77, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 38, 38, 64)       0
 2D)

 dropout_5 (Dropout)         (None, 38, 38, 64)        0

 conv2d_5 (Conv2D)           (None, 36, 36, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 18, 18, 128)      0
 2D)

 dropout_6 (Dropout)         (None, 18, 18, 128)       0

 conv2d_6 (Conv2D)           (None, 16, 16, 128)       147584

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 dropout_7 (Dropout)         (None, 8, 8, 128)         0

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 128)               1048704

 dropout_8 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 1,290,181
Trainable params: 1,290,181
Non-trainable params: 0
_________________________________________________________________
None
```

Checking the test accuracy for both models

In [ ]:

```
from sklearn.metrics import accuracy_score
base_dir = '/content/drive/My Drive/assignment'
name_rock = 'model_final5_rock'
name_flower = 'model_final5_flower'
model_rock = load_model_from_file(base_dir, name_rock, extension='.h5')
model_flower = load_model_from_file(base_dir, name_flower, extension='.h5')

yr_pred = model_rock.predict(Xr_test)
yf_pred = model_flower.predict(Xf_test)
score_r = accuracy_score(np.argmax(yr_test, axis=1), np.argmax(yr_pred, axis=1))
score_f = accuracy_score(np.argmax(yf_test, axis=1), np.argmax(yf_pred, axis=1))
print("Test accuracy rock paper scissors model: {:.2f}".format(score_r))
print("Test accuracy flower model: {:.2f}".format(score_f))
```

```
12/12 [==============================] - 0s 15ms/step
23/23 [==============================] - 0s 13ms/step
Test accuracy rock paper scissors model: 0.84
Test accuracy flower model: 0.74
```

## 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.

In [ ]:

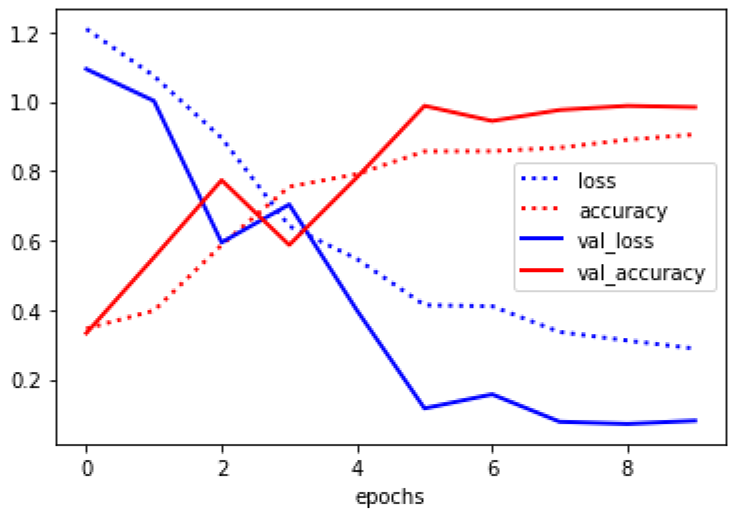
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Function to augment the rock data
def augment_rock_data(X_train, y_train, X_valid, y_valid):
  """ Augments the data and returns a generator and the validation data and labels
  """
  generator = ImageDataGenerator(
      width_shift_range=0.2,
      height_shift_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True
  ).flow(X_train, y_train)
  return generator, X_valid, y_valid

# Function to augment the flower data
def augment_flower_data(X_train, y_train, X_valid, y_valid):
  """ Augments the data and returns a generator and the validation data and labels
  """
  generator = ImageDataGenerator(
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True
  ).flow(X_train, y_train)
  return generator, X_valid, y_valid

# Augment rock data
def augment_data_rock():
  return augment_rock_data(Xr_train, yr_train, Xr_valid, yr_valid)

# Augment flower data
def augment_data_flower():
  return augment_flower_data(Xf_train, yf_train, Xf_valid, yf_valid)
```

In [ ]:

```
# Run rock paper scissor model with augmented data
run_evaluation("augmented_model_final5_rock", build_r_model, augment_data_rock(), base_di
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 12s 181ms/step - loss: 1.2095 - accuracy: 0.3464
 - val_loss: 1.0940 - val_accuracy: 0.3333
Epoch 2/10
60/60 [==============================] - 11s 175ms/step - loss: 1.0740 - accuracy: 0.3982
 - val_loss: 1.0023 - val_accuracy: 0.5516
Epoch 3/10
60/60 [==============================] - 10s 174ms/step - loss: 0.8954 - accuracy: 0.5872
 - val_loss: 0.5946 - val_accuracy: 0.7738
Epoch 4/10
60/60 [==============================] - 11s 175ms/step - loss: 0.6418 - accuracy: 0.7552
```

```
 - val_loss: 0.7040 - val_accuracy: 0.5873
Epoch 5/10
60/60 [==============================] - 11s 176ms/step - loss: 0.5469 - accuracy: 0.7912
 - val_loss: 0.3994 - val_accuracy: 0.7817
Epoch 6/10
60/60 [==============================] - 11s 175ms/step - loss: 0.4140 - accuracy: 0.8570
 - val_loss: 0.1165 - val_accuracy: 0.9881
Epoch 7/10
60/60 [==============================] - 12s 195ms/step - loss: 0.4107 - accuracy: 0.8575
 - val_loss: 0.1571 - val_accuracy: 0.9444
Epoch 8/10
60/60 [==============================] - 10s 174ms/step - loss: 0.3365 - accuracy: 0.8674
 - val_loss: 0.0780 - val_accuracy: 0.9762
Epoch 9/10
60/60 [==============================] - 11s 188ms/step - loss: 0.3119 - accuracy: 0.8904
 - val_loss: 0.0718 - val_accuracy: 0.9881
Epoch 10/10
60/60 [==============================] - 10s 174ms/step - loss: 0.2888 - accuracy: 0.9055
 - val_loss: 0.0811 - val_accuracy: 0.9841
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 98.81%
```




```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 158, 158, 32)      896

 max_pooling2d_14 (MaxPoolin  (None, 79, 79, 32)       0
 g2D)

 dropout_18 (Dropout)        (None, 79, 79, 32)        0

 conv2d_15 (Conv2D)          (None, 77, 77, 64)        18496

 max_pooling2d_15 (MaxPoolin  (None, 38, 38, 64)       0
 g2D)

 dropout_19 (Dropout)        (None, 38, 38, 64)        0

 conv2d_16 (Conv2D)          (None, 36, 36, 64)        36928

 max_pooling2d_16 (MaxPoolin  (None, 18, 18, 64)       0
 g2D)

 dropout_20 (Dropout)        (None, 18, 18, 64)        0

 flatten_4 (Flatten)         (None, 20736)             0
```

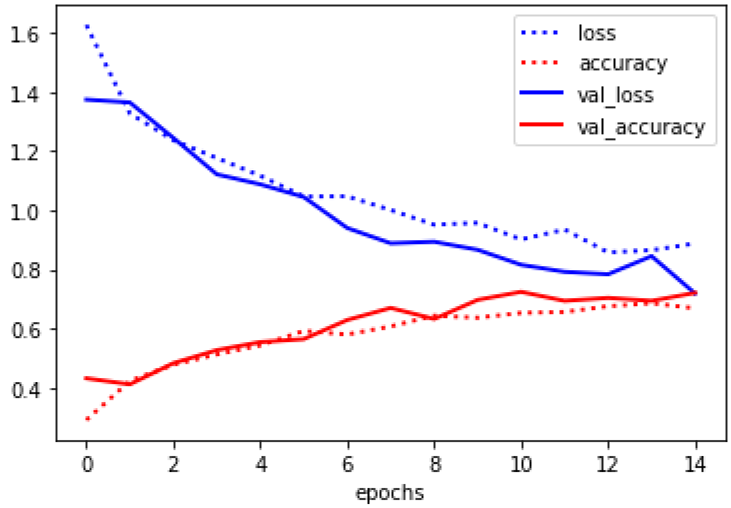
```
 dense_8 (Dense)             (None, 64)                1327168

 dropout_21 (Dropout)        (None, 64)                0

 dense_9 (Dense)             (None, 3)                 195

=================================================================
Total params: 1,383,683
Trainable params: 1,383,683
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
# Run flower model with augmented data
run_evaluation("augmented_model_final5_flower", build_f_model, augment_data_flower(), base
               train=True, generator=True, epochs=15, batch_size=None)
```

```
Training the model
Epoch 1/15
60/60 [==============================] - 12s 178ms/step - loss: 1.6274 - accuracy: 0.2912
 - val_loss: 1.3749 - val_accuracy: 0.4320
Epoch 2/15
60/60 [==============================] - 11s 177ms/step - loss: 1.3281 - accuracy: 0.4229
 - val_loss: 1.3652 - val_accuracy: 0.4116
Epoch 3/15
60/60 [==============================] - 11s 176ms/step - loss: 1.2374 - accuracy: 0.4780
 - val_loss: 1.2455 - val_accuracy: 0.4830
Epoch 4/15
60/60 [==============================] - 11s 176ms/step - loss: 1.1771 - accuracy: 0.5126
 - val_loss: 1.1220 - val_accuracy: 0.5272
Epoch 5/15
60/60 [==============================] - 11s 179ms/step - loss: 1.1174 - accuracy: 0.5435
 - val_loss: 1.0883 - val_accuracy: 0.5544
Epoch 6/15
60/60 [==============================] - 11s 177ms/step - loss: 1.0468 - accuracy: 0.5922
 - val_loss: 1.0453 - val_accuracy: 0.5646
Epoch 7/15
60/60 [==============================] - 11s 178ms/step - loss: 1.0477 - accuracy: 0.5803
 - val_loss: 0.9406 - val_accuracy: 0.6293
Epoch 8/15
60/60 [==============================] - 11s 176ms/step - loss: 1.0011 - accuracy: 0.6076
 - val_loss: 0.8888 - val_accuracy: 0.6701
Epoch 9/15
60/60 [==============================] - 11s 175ms/step - loss: 0.9515 - accuracy: 0.6438
 - val_loss: 0.8940 - val_accuracy: 0.6327
Epoch 10/15
60/60 [==============================] - 12s 197ms/step - loss: 0.9577 - accuracy: 0.6369
 - val_loss: 0.8667 - val_accuracy: 0.6973
Epoch 11/15
60/60 [==============================] - 11s 177ms/step - loss: 0.9016 - accuracy: 0.6531
 - val_loss: 0.8160 - val_accuracy: 0.7245
Epoch 12/15
60/60 [==============================] - 11s 177ms/step - loss: 0.9353 - accuracy: 0.6563
 - val_loss: 0.7922 - val_accuracy: 0.6939
Epoch 13/15
60/60 [==============================] - 10s 173ms/step - loss: 0.8580 - accuracy: 0.6758
 - val_loss: 0.7839 - val_accuracy: 0.7041
Epoch 14/15
60/60 [==============================] - 11s 176ms/step - loss: 0.8655 - accuracy: 0.6859
 - val_loss: 0.8454 - val_accuracy: 0.6939
Epoch 15/15
60/60 [==============================] - 10s 173ms/step - loss: 0.8879 - accuracy: 0.6689
 - val_loss: 0.7189 - val_accuracy: 0.7211
Saving to file
```

```
Model stored in /content/drive/My Drive/assignment
Max val score: 72.45%
```




```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 158, 158, 32)      896       
                                                                 
 max_pooling2d_8 (MaxPooling  (None, 79, 79, 32)       0         
 2D)                                                             
                                                                 
 dropout_10 (Dropout)        (None, 79, 79, 32)        0         
                                                                 
 conv2d_9 (Conv2D)           (None, 77, 77, 64)        18496     
                                                                 
 max_pooling2d_9 (MaxPooling  (None, 38, 38, 64)       0         
 2D)                                                             
                                                                 
 dropout_11 (Dropout)        (None, 38, 38, 64)        0         
                                                                 
 conv2d_10 (Conv2D)          (None, 36, 36, 128)       73856     
                                                                 
 max_pooling2d_10 (MaxPoolin  (None, 18, 18, 128)      0         
 g2D)                                                            
                                                                 
 dropout_12 (Dropout)        (None, 18, 18, 128)       0         
                                                                 
 conv2d_11 (Conv2D)          (None, 16, 16, 128)       147584    
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 8, 8, 128)        0         
 g2D)                                                            
                                                                 
 dropout_13 (Dropout)        (None, 8, 8, 128)         0         
                                                                 
 flatten_2 (Flatten)         (None, 8192)              0         
                                                                 
 dense_4 (Dense)             (None, 128)               1048704   
                                                                 
 dropout_14 (Dropout)        (None, 128)               0         
                                                                 
 dense_5 (Dense)             (None, 5)                 645       
                                                                 
=================================================================
Total params: 1,290,181
Trainable params: 1,290,181
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.

### For the rock dataset (2 cells)

In [ ]:

```
#loading model
base_dir = '/content/drive/My Drive/assignment'
name = 'augmented_model_final5_rock'
rock_model = load_model_from_file(base_dir, name, extension='.h5')
```

In [ ]:

```
#To calculate accuracy
yr_pred_test = rock_model.predict(Xr_test)
yr_pred_train = rock_model.predict(Xr_train)
yr_pred_valid = rock_model.predict(Xr_valid)

#predict classes for the test set
rock_model_misclassified_samples = np.nonzero(np.argmax(yr_test, axis=1) != np.argmax(yr_p

rock_misclassified = []
paper_misclassified = []
scissors_misclassified = []

for i in range (0, len(rock_model_misclassified_samples)):
  if class_names_r[np.argmax(yr_test[rock_model_misclassified_samples[i]])] == "rock":
    rock_misclassified.append(rock_model_misclassified_samples[i])
  if class_names_r[np.argmax(yr_test[rock_model_misclassified_samples[i]])] == "paper":
    paper_misclassified.append(rock_model_misclassified_samples[i])
  if class_names_r[np.argmax(yr_test[rock_model_misclassified_samples[i]])] == "scissors":
    scissors_misclassified.append(rock_model_misclassified_samples[i])

from sklearn.metrics import accuracy_score
print("Accuracy score on train set: " + str(round(accuracy_score(np.argmax(yr_train, axis=
print("Accuracy score on test set: " + str(round(accuracy_score(np.argmax(yr_test, axis=1)
print("Accuracy score on validation set: " + str(round(accuracy_score(np.argmax(yr_valid,
```

```
12/12 [==============================] - 0s 26ms/step
71/71 [==============================] - 1s 14ms/step
8/8 [==============================] - 0s 12ms/step
Accuracy score on train set: 0.98
Accuracy score on test set: 0.93
Accuracy score on validation set: 0.98
```

### For the flower dataset (2 cells)

In [ ]:

```
#loading model
base_dir = '/content/drive/My Drive/assignment'
name = 'augmented_model_final5_flower'
flower_model = load_model_from_file(base_dir, name, extension='.h5')
```

In [ ]:

```
#To calculate accuracy
yf_pred_test = flower_model.predict(Xf_test)
yf_pred_train = flower_model.predict(Xf_train)
```

```
yf_pred_valid = flower_model.predict(Xf_valid)
#predict classes for the test set
flower_model_misclassified_samples = np.nonzero(np.argmax(yf_test, axis=1) != np.argmax(yf


dandelion_misclassified = []
daisy_misclassified = []
tulips_misclassified = []
sunflowers_misclassified = []
roses_misclassified = []

for i in range (0, len(flower_model_misclassified_samples)):
  if class_names_f[np.argmax(yf_test[flower_model_misclassified_samples[i]])] == "dandelic
    dandelion_misclassified.append(flower_model_misclassified_samples[i])
  if class_names_f[np.argmax(yf_test[flower_model_misclassified_samples[i]])] == "daisy":
    daisy_misclassified.append(flower_model_misclassified_samples[i])
  if class_names_f[np.argmax(yf_test[flower_model_misclassified_samples[i]])] == "tulips":
    tulips_misclassified.append(flower_model_misclassified_samples[i])
  if class_names_f[np.argmax(yf_test[flower_model_misclassified_samples[i]])] == "sunflowe
    sunflowers_misclassified.append(flower_model_misclassified_samples[i])
  if class_names_f[np.argmax(yf_test[flower_model_misclassified_samples[i]])] == "roses":
    roses_misclassified.append(flower_model_misclassified_samples[i])

from sklearn.metrics import accuracy_score
print("Accuracy score on train set: " + str(round(accuracy_score(np.argmax(yf_train, axis=
print("Accuracy score on test set: " + str(round(accuracy_score(np.argmax(yf_test, axis=1)
print("Accuracy score on validation set: " + str(round(accuracy_score(np.argmax(yf_valid,
```

```
23/23 [==============================] - 0s 14ms/step
83/83 [==============================] - 1s 13ms/step
10/10 [==============================] - 0s 13ms/step
Accuracy score on train set: 0.75
Accuracy score on test set: 0.73
Accuracy score on validation set: 0.721
```

## 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 occurences 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)?

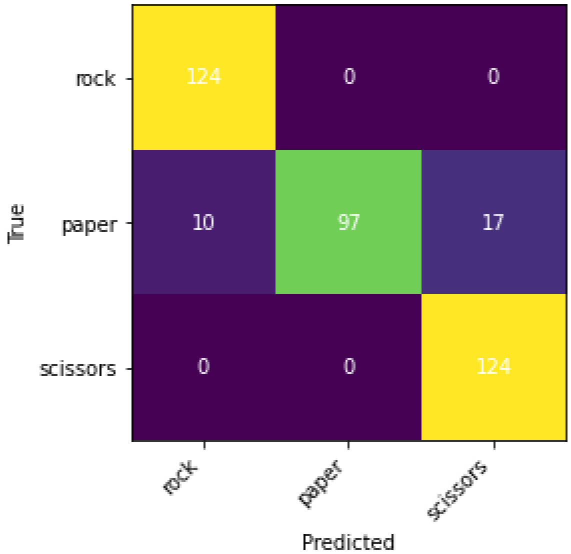
**Note:** If, for some classes, you cannot find 5 misclassifications, simply plot all misclassifications for that class.

### For the rock dataset (2 cells)

In [ ]:

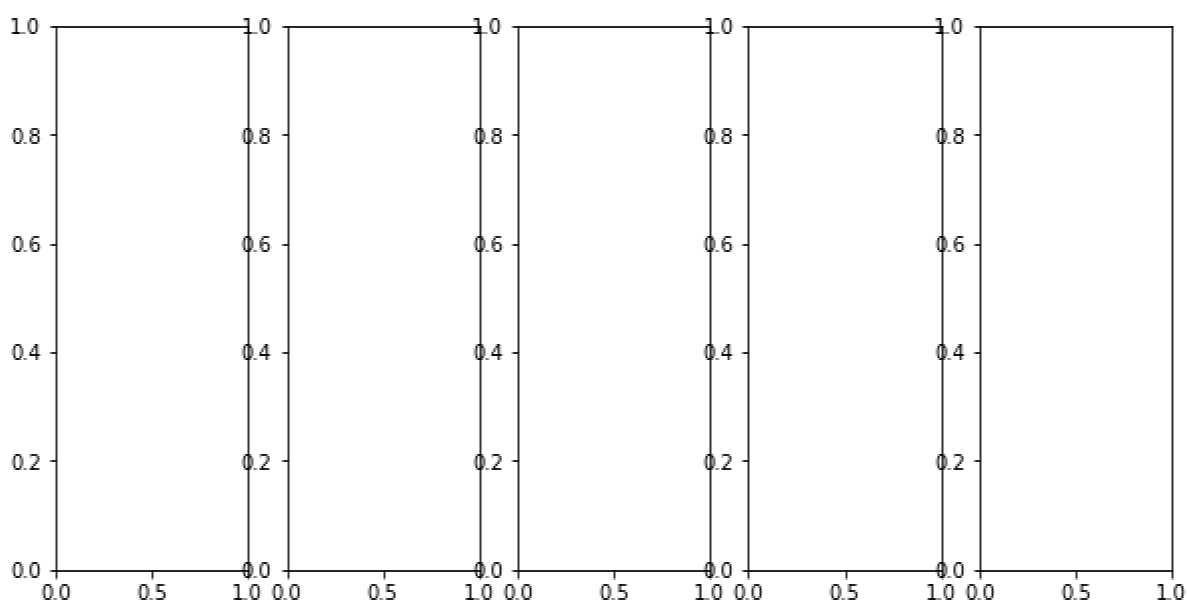
```
#Plot the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(np.argmax(yr_test, axis=1),np.argmax(yr_pred_test, axis=1))
fig, ax = plt.subplots()
im = ax.imshow(cm)
ax.set_xticks(np.arange(3)), ax.set_yticks(np.arange(3))
ax.set_xticklabels(list(class_names_r), rotation=45, ha="right")
ax.set_yticklabels(list(class_names_r))
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")
```




In [ ]:

```
#Analyze misclassified cases
number_of_examples = 5
misclassified_classes_r = [rock_misclassified, paper_misclassified, scissors_misclassifie

for i in range(0, len(misclassified_classes_r)):
  fig, axes = plt.subplots(1, number_of_examples,  figsize=(10, 5))
  for nr, i in enumerate(misclassified_classes_r[i][:number_of_examples]):
      axes[nr].imshow(Xr_test[i])
      axes[nr].set_xlabel("Predicted: %s,\n Actual : %s" % (class_names_r[np.argmax(yr_pre
      axes[nr].set_xticks(()), axes[nr].set_yticks(())

  plt.show();
```



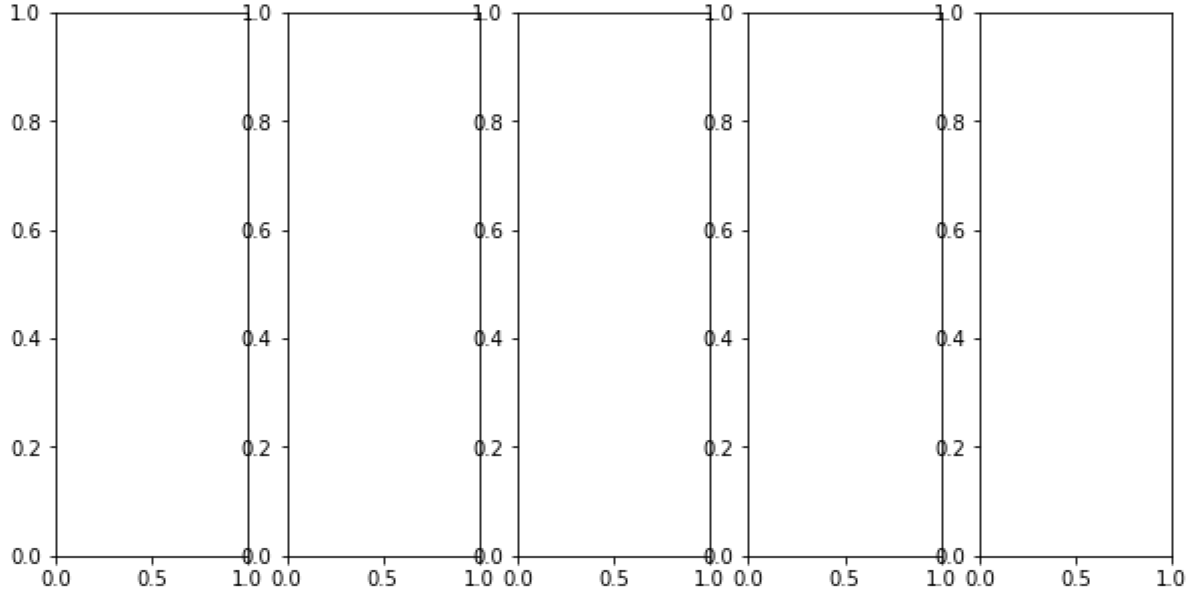
Predicted: scissors, Actual : paper

Predicted: rock, Actual : paper

Predicted: scissors, Actual : paper

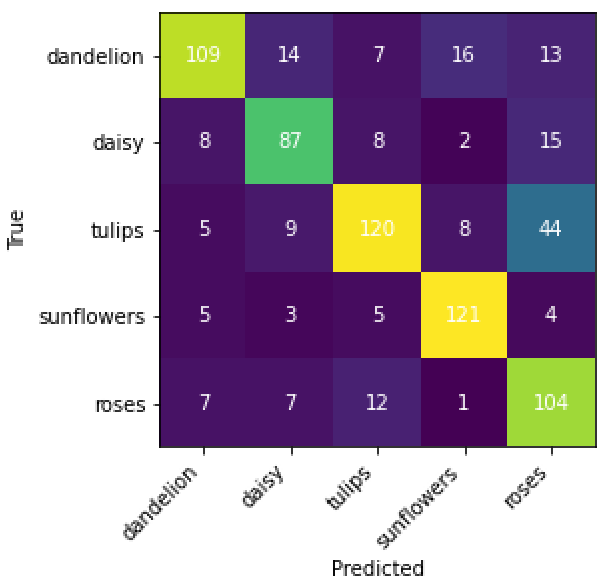
Predicted: rock, Actual : paper

Predicted: scissors, Actual : paper



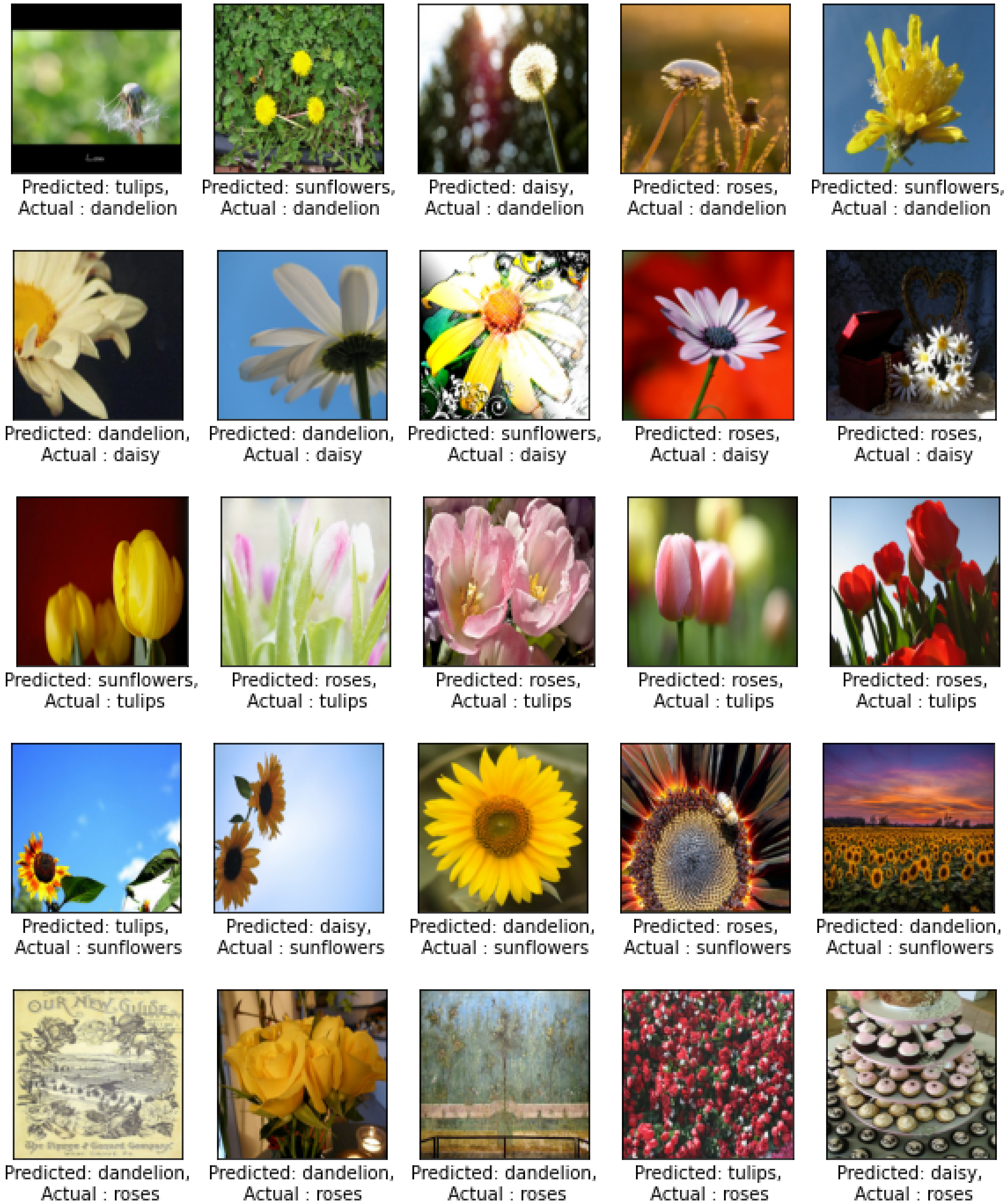

for the flower dataset (2 cells)

In [ ]:
```
#Plot the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(np.argmax(yf_test, axis=1),np.argmax(yf_pred_test, 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(class_names_f), rotation=45, ha="right")
ax.set_yticklabels(list(class_names_f))
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")
```




In [ ]:
```
#Analyze misclassified cases
number_of_examples = 5

misclassified_classes_f = [dandelion_misclassified, daisy_misclassified, tulips_misclassi

for i in range(0, len(misclassified_classes_f)):
  fig, axes = plt.subplots(1, number_of_examples,  figsize=(10, 5))
  for nr, i in enumerate(misclassified_classes_f[i][:number_of_examples]):
      axes[nr].imshow(Xf_test[i])
```

```
    axes[nr].set_xlabel("Predicted: %s,\n Actual : %s" % (class_names_f[np.argmax(yf_pre
    axes[nr].set_xticks(()), axes[nr].set_yticks(())

plt.show();
```




# 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.

In [ ]:

```
# Import MobileNetV2 model
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
conv_base = MobileNetV2(weights="imagenet",
                        include_top=False,
                        input_shape=(IMG_SIZE, IMG_SIZE, 3))
conv_base.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilen
et_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
9406464/9406464 [==============================] - 0s 0us/step
Model: "mobilenetv2_1.00_160"
__________________________________________________________________________________________
________
 Layer (type)                Output Shape         Param #     Connected to

==========================================================================================
========
 input_1 (InputLayer)        [(None, 160, 160, 3  0           []
                             )]


 Conv1 (Conv2D)              (None, 80, 80, 32)   864         ['input_1[0][0]']


 bn_Conv1 (BatchNormalization)  (None, 80, 80, 32)   128         ['Conv1[0][0]']


 Conv1_relu (ReLU)           (None, 80, 80, 32)   0           ['bn_Conv1[0][0]']


 expanded_conv_depthwise (Depth  (None, 80, 80, 32)  288         ['Conv1_relu[0][0]']

 wiseConv2D)


 expanded_conv_depthwise_BN (Ba  (None, 80, 80, 32)  128         ['expanded_conv_depthwise
[0][0]']
 tchNormalization)


 expanded_conv_depthwise_relu (  (None, 80, 80, 32)  0           ['expanded_conv_depthwise
_BN[0][0
 ReLU)                                                            ]']


 expanded_conv_project (Conv2D)  (None, 80, 80, 16)  512         ['expanded_conv_depthwise
_relu[0]
```

```
                                                                  [0]']

 expanded_conv_project_BN (Batc  (None, 80, 80, 16)  64          ['expanded_conv_project
[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 80, 80, 96)   1536        ['expanded_conv_project_B
N[0][0]'
                                                                  ]

 block_1_expand_BN (BatchNormal  (None, 80, 80, 96)  384         ['block_1_expand[0][0]']

 ization)

 block_1_expand_relu (ReLU)     (None, 80, 80, 96)   0           ['block_1_expand_BN[0]
[0]']

 block_1_pad (ZeroPadding2D)    (None, 81, 81, 96)   0           ['block_1_expand_relu[0]
[0]']

 block_1_depthwise (DepthwiseCo  (None, 40, 40, 96)  864         ['block_1_pad[0][0]']

 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 40, 40, 96)  384         ['block_1_depthwise[0]
[0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 40, 40, 96)   0           ['block_1_depthwise_BN[0]
[0]']

 block_1_project (Conv2D)       (None, 40, 40, 24)   2304        ['block_1_depthwise_relu
[0][0]']

 block_1_project_BN (BatchNorma  (None, 40, 40, 24)  96          ['block_1_project[0][0]']

 lization)

 block_2_expand (Conv2D)        (None, 40, 40, 144)  3456        ['block_1_project_BN[0]
[0]']

 block_2_expand_BN (BatchNormal  (None, 40, 40, 144)  576        ['block_2_expand[0][0]']

 ization)
```

```
 block_2_expand_relu (ReLU)     (None, 40, 40, 144)  0           ['block_2_expand_BN[0]
[0]']


 block_2_depthwise (DepthwiseCo  (None, 40, 40, 144)  1296       ['block_2_expand_relu[0]
[0]']
 nv2D)



 block_2_depthwise_BN (BatchNor  (None, 40, 40, 144)  576        ['block_2_depthwise[0]
[0]']
 malization)



 block_2_depthwise_relu (ReLU)  (None, 40, 40, 144)  0           ['block_2_depthwise_BN[0]
[0]']


 block_2_project (Conv2D)       (None, 40, 40, 24)   3456        ['block_2_depthwise_relu
[0][0]']


 block_2_project_BN (BatchNorma  (None, 40, 40, 24)  96          ['block_2_project[0][0]']

 lization)



 block_2_add (Add)              (None, 40, 40, 24)   0           ['block_1_project_BN[0]
[0]',
                                                                  'block_2_project_BN[0]
[0]']


 block_3_expand (Conv2D)        (None, 40, 40, 144)  3456        ['block_2_add[0][0]']



 block_3_expand_BN (BatchNormal  (None, 40, 40, 144)  576        ['block_3_expand[0][0]']

 ization)



 block_3_expand_relu (ReLU)     (None, 40, 40, 144)  0           ['block_3_expand_BN[0]
[0]']


 block_3_pad (ZeroPadding2D)    (None, 41, 41, 144)  0           ['block_3_expand_relu[0]
[0]']


 block_3_depthwise (DepthwiseCo  (None, 20, 20, 144)  1296       ['block_3_pad[0][0]']

 nv2D)



 block_3_depthwise_BN (BatchNor  (None, 20, 20, 144)  576        ['block_3_depthwise[0]
[0]']
 malization)
```

```
 block_3_depthwise_relu (ReLU)  (None, 20, 20, 144)  0           ['block_3_depthwise_BN[0]
[0]']

 block_3_project (Conv2D)       (None, 20, 20, 32)   4608        ['block_3_depthwise_relu
[0][0]']

 block_3_project_BN (BatchNorma  (None, 20, 20, 32)  128         ['block_3_project[0][0]']

 lization)


 block_4_expand (Conv2D)        (None, 20, 20, 192)  6144        ['block_3_project_BN[0]
[0]']

 block_4_expand_BN (BatchNormal  (None, 20, 20, 192)  768        ['block_4_expand[0][0]']

 ization)


 block_4_expand_relu (ReLU)     (None, 20, 20, 192)  0           ['block_4_expand_BN[0]
[0]']

 block_4_depthwise (DepthwiseCo  (None, 20, 20, 192)  1728       ['block_4_expand_relu[0]
[0]']
 nv2D)


 block_4_depthwise_BN (BatchNor  (None, 20, 20, 192)  768        ['block_4_depthwise[0]
[0]']
 malization)


 block_4_depthwise_relu (ReLU)  (None, 20, 20, 192)  0           ['block_4_depthwise_BN[0]
[0]']

 block_4_project (Conv2D)       (None, 20, 20, 32)   6144        ['block_4_depthwise_relu
[0][0]']

 block_4_project_BN (BatchNorma  (None, 20, 20, 32)  128         ['block_4_project[0][0]']

 lization)


 block_4_add (Add)              (None, 20, 20, 32)   0           ['block_3_project_BN[0]
[0]',
                                                                  'block_4_project_BN[0]
[0]']

 block_5_expand (Conv2D)        (None, 20, 20, 192)  6144        ['block_4_add[0][0]']


 block_5_expand_BN (BatchNormal  (None, 20, 20, 192)  768        ['block_5_expand[0][0]']
```

```
 ization)


 block_5_expand_relu (ReLU)     (None, 20, 20, 192)  0           ['block_5_expand_BN[0]
[0]']

 block_5_depthwise (DepthwiseCo  (None, 20, 20, 192)  1728       ['block_5_expand_relu[0]
[0]']
 nv2D)


 block_5_depthwise_BN (BatchNor  (None, 20, 20, 192)  768        ['block_5_depthwise[0]
[0]']
 malization)


 block_5_depthwise_relu (ReLU)  (None, 20, 20, 192)  0           ['block_5_depthwise_BN[0]
[0]']

 block_5_project (Conv2D)       (None, 20, 20, 32)   6144        ['block_5_depthwise_relu
[0][0]']

 block_5_project_BN (BatchNorma  (None, 20, 20, 32)  128         ['block_5_project[0][0]']

 lization)


 block_5_add (Add)              (None, 20, 20, 32)   0           ['block_4_add[0][0]',

                                                                  'block_5_project_BN[0]
[0]']

 block_6_expand (Conv2D)        (None, 20, 20, 192)  6144        ['block_5_add[0][0]']


 block_6_expand_BN (BatchNormal  (None, 20, 20, 192)  768        ['block_6_expand[0][0]']

 ization)


 block_6_expand_relu (ReLU)     (None, 20, 20, 192)  0           ['block_6_expand_BN[0]
[0]']

 block_6_pad (ZeroPadding2D)    (None, 21, 21, 192)  0           ['block_6_expand_relu[0]
[0]']

 block_6_depthwise (DepthwiseCo  (None, 10, 10, 192)  1728       ['block_6_pad[0][0]']

 nv2D)


 block_6_depthwise_BN (BatchNor  (None, 10, 10, 192)  768        ['block_6_depthwise[0]
[0]']
```

```
 malization)


 block_6_depthwise_relu (ReLU)  (None, 10, 10, 192)  0           ['block_6_depthwise_BN[0]
[0]']


 block_6_project (Conv2D)       (None, 10, 10, 64)   12288       ['block_6_depthwise_relu
[0][0]']


 block_6_project_BN (BatchNorma  (None, 10, 10, 64)  256         ['block_6_project[0][0]']

 lization)



 block_7_expand (Conv2D)        (None, 10, 10, 384)  24576       ['block_6_project_BN[0]
[0]']


 block_7_expand_BN (BatchNormal  (None, 10, 10, 384)  1536       ['block_7_expand[0][0]']

 ization)



 block_7_expand_relu (ReLU)     (None, 10, 10, 384)  0           ['block_7_expand_BN[0]
[0]']


 block_7_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456       ['block_7_expand_relu[0]
[0]']
 nv2D)



 block_7_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536       ['block_7_depthwise[0]
[0]']
 malization)



 block_7_depthwise_relu (ReLU)  (None, 10, 10, 384)  0           ['block_7_depthwise_BN[0]
[0]']


 block_7_project (Conv2D)       (None, 10, 10, 64)   24576       ['block_7_depthwise_relu
[0][0]']


 block_7_project_BN (BatchNorma  (None, 10, 10, 64)  256         ['block_7_project[0][0]']

 lization)



 block_7_add (Add)              (None, 10, 10, 64)   0           ['block_6_project_BN[0]
[0]',

                                                                  'block_7_project_BN[0]
[0]']


 block_8_expand (Conv2D)        (None, 10, 10, 384)  24576       ['block_7_add[0][0]']
```

```
 block_8_expand_BN (BatchNormal  (None, 10, 10, 384)  1536       ['block_8_expand[0][0]']

 ization)


 block_8_expand_relu (ReLU)     (None, 10, 10, 384)  0          ['block_8_expand_BN[0]
[0]']

 block_8_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456       ['block_8_expand_relu[0]
[0]']
 nv2D)


 block_8_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536       ['block_8_depthwise[0]
[0]']
 malization)


 block_8_depthwise_relu (ReLU)  (None, 10, 10, 384)  0          ['block_8_depthwise_BN[0]
[0]']

 block_8_project (Conv2D)       (None, 10, 10, 64)   24576      ['block_8_depthwise_relu
[0][0]']

 block_8_project_BN (BatchNorma  (None, 10, 10, 64)  256        ['block_8_project[0][0]']

 lization)


 block_8_add (Add)              (None, 10, 10, 64)   0          ['block_7_add[0][0]',

                                                                 'block_8_project_BN[0]
[0]']

 block_9_expand (Conv2D)        (None, 10, 10, 384)  24576      ['block_8_add[0][0]']


 block_9_expand_BN (BatchNormal  (None, 10, 10, 384)  1536       ['block_9_expand[0][0]']

 ization)


 block_9_expand_relu (ReLU)     (None, 10, 10, 384)  0          ['block_9_expand_BN[0]
[0]']

 block_9_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456       ['block_9_expand_relu[0]
[0]']
 nv2D)


 block_9_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536       ['block_9_depthwise[0]
[0]']
```

```
 malization)


 block_9_depthwise_relu (ReLU)  (None, 10, 10, 384)  0           ['block_9_depthwise_BN[0]
[0]']

 block_9_project (Conv2D)       (None, 10, 10, 64)   24576       ['block_9_depthwise_relu
[0][0]']

 block_9_project_BN (BatchNorma  (None, 10, 10, 64)  256         ['block_9_project[0][0]']

 lization)


 block_9_add (Add)              (None, 10, 10, 64)   0           ['block_8_add[0][0]',

                                                                  'block_9_project_BN[0]
[0]']

 block_10_expand (Conv2D)       (None, 10, 10, 384)  24576       ['block_9_add[0][0]']


 block_10_expand_BN (BatchNorma  (None, 10, 10, 384)  1536       ['block_10_expand[0][0]']

 lization)


 block_10_expand_relu (ReLU)    (None, 10, 10, 384)  0           ['block_10_expand_BN[0]
[0]']

 block_10_depthwise (DepthwiseC  (None, 10, 10, 384)  3456       ['block_10_expand_relu[0]
[0]']
 onv2D)


 block_10_depthwise_BN (BatchNo  (None, 10, 10, 384)  1536       ['block_10_depthwise[0]
[0]']
 rmalization)


 block_10_depthwise_relu (ReLU)  (None, 10, 10, 384)  0          ['block_10_depthwise_BN
[0][0]']

 block_10_project (Conv2D)      (None, 10, 10, 96)   36864       ['block_10_depthwise_relu
[0][0]']

 block_10_project_BN (BatchNorm  (None, 10, 10, 96)  384         ['block_10_project[0]
[0]']
 alization)


 block_11_expand (Conv2D)       (None, 10, 10, 576)  55296       ['block_10_project_BN[0]
[0]']
```

```
 block_11_expand_BN (BatchNorma  (None, 10, 10, 576)  2304       ['block_11_expand[0][0]']

 lization)



 block_11_expand_relu (ReLU)    (None, 10, 10, 576)  0          ['block_11_expand_BN[0]
[0]']

 block_11_depthwise (DepthwiseC  (None, 10, 10, 576)  5184       ['block_11_expand_relu[0]
[0]']
 onv2D)



 block_11_depthwise_BN (BatchNo  (None, 10, 10, 576)  2304       ['block_11_depthwise[0]
[0]']
 rmalization)



 block_11_depthwise_relu (ReLU)  (None, 10, 10, 576)  0          ['block_11_depthwise_BN
[0][0]']

 block_11_project (Conv2D)      (None, 10, 10, 96)   55296      ['block_11_depthwise_relu
[0][0]']

 block_11_project_BN (BatchNorm  (None, 10, 10, 96)  384        ['block_11_project[0]
[0]']
 alization)



 block_11_add (Add)             (None, 10, 10, 96)   0          ['block_10_project_BN[0]
[0]',
                                                                 'block_11_project_BN[0]
[0]']

 block_12_expand (Conv2D)       (None, 10, 10, 576)  55296      ['block_11_add[0][0]']



 block_12_expand_BN (BatchNorma  (None, 10, 10, 576)  2304       ['block_12_expand[0][0]']

 lization)



 block_12_expand_relu (ReLU)    (None, 10, 10, 576)  0          ['block_12_expand_BN[0]
[0]']

 block_12_depthwise (DepthwiseC  (None, 10, 10, 576)  5184       ['block_12_expand_relu[0]
[0]']
 onv2D)



 block_12_depthwise_BN (BatchNo  (None, 10, 10, 576)  2304       ['block_12_depthwise[0]
[0]']
```

```
 rmalization)


 block_12_depthwise_relu (ReLU)  (None, 10, 10, 576)  0           ['block_12_depthwise_BN
[0][0]']

 block_12_project (Conv2D)      (None, 10, 10, 96)   55296       ['block_12_depthwise_relu
[0][0]']

 block_12_project_BN (BatchNorm  (None, 10, 10, 96)  384         ['block_12_project[0]
[0]']
 alization)


 block_12_add (Add)             (None, 10, 10, 96)   0           ['block_11_add[0][0]',

                                                                  'block_12_project_BN[0]
[0]']

 block_13_expand (Conv2D)       (None, 10, 10, 576)  55296       ['block_12_add[0][0]']


 block_13_expand_BN (BatchNorma  (None, 10, 10, 576)  2304       ['block_13_expand[0][0]']

 lization)


 block_13_expand_relu (ReLU)    (None, 10, 10, 576)  0           ['block_13_expand_BN[0]
[0]']

 block_13_pad (ZeroPadding2D)   (None, 11, 11, 576)  0           ['block_13_expand_relu[0]
[0]']

 block_13_depthwise (DepthwiseC  (None, 5, 5, 576)   5184        ['block_13_pad[0][0]']

 onv2D)


 block_13_depthwise_BN (BatchNo  (None, 5, 5, 576)   2304        ['block_13_depthwise[0]
[0]']
 rmalization)


 block_13_depthwise_relu (ReLU)  (None, 5, 5, 576)   0           ['block_13_depthwise_BN
[0][0]']

 block_13_project (Conv2D)      (None, 5, 5, 160)    92160       ['block_13_depthwise_relu
[0][0]']

 block_13_project_BN (BatchNorm  (None, 5, 5, 160)   640         ['block_13_project[0]
[0]']
 alization)
```

```
 block_14_expand (Conv2D)       (None, 5, 5, 960)    153600      ['block_13_project_BN[0]
[0]']

 block_14_expand_BN (BatchNorma  (None, 5, 5, 960)   3840        ['block_14_expand[0][0]']

 lization)


 block_14_expand_relu (ReLU)    (None, 5, 5, 960)    0           ['block_14_expand_BN[0]
[0]']

 block_14_depthwise (DepthwiseC  (None, 5, 5, 960)   8640        ['block_14_expand_relu[0]
[0]']
 onv2D)


 block_14_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840        ['block_14_depthwise[0]
[0]']
 rmalization)


 block_14_depthwise_relu (ReLU)  (None, 5, 5, 960)   0           ['block_14_depthwise_BN
[0][0]']

 block_14_project (Conv2D)      (None, 5, 5, 160)    153600      ['block_14_depthwise_relu
[0][0]']

 block_14_project_BN (BatchNorm  (None, 5, 5, 160)   640         ['block_14_project[0]
[0]']
 alization)


 block_14_add (Add)             (None, 5, 5, 160)    0           ['block_13_project_BN[0]
[0]',
                                                                  'block_14_project_BN[0]
[0]']

 block_15_expand (Conv2D)       (None, 5, 5, 960)    153600      ['block_14_add[0][0]']


 block_15_expand_BN (BatchNorma  (None, 5, 5, 960)   3840        ['block_15_expand[0][0]']

 lization)


 block_15_expand_relu (ReLU)    (None, 5, 5, 960)    0           ['block_15_expand_BN[0]
[0]']

 block_15_depthwise (DepthwiseC  (None, 5, 5, 960)   8640        ['block_15_expand_relu[0]
[0]']
 onv2D)
```

```
 block_15_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840        ['block_15_depthwise[0]
[0]']
 rmalization)


 block_15_depthwise_relu (ReLU)  (None, 5, 5, 960)   0           ['block_15_depthwise_BN
[0][0]']

 block_15_project (Conv2D)      (None, 5, 5, 160)    153600      ['block_15_depthwise_relu
[0][0]']

 block_15_project_BN (BatchNorm  (None, 5, 5, 160)   640         ['block_15_project[0]
[0]']
 alization)


 block_15_add (Add)             (None, 5, 5, 160)    0           ['block_14_add[0][0]',

                                                                  'block_15_project_BN[0]
[0]']

 block_16_expand (Conv2D)       (None, 5, 5, 960)    153600      ['block_15_add[0][0]']


 block_16_expand_BN (BatchNorma  (None, 5, 5, 960)   3840        ['block_16_expand[0][0]']

 lization)


 block_16_expand_relu (ReLU)    (None, 5, 5, 960)    0           ['block_16_expand_BN[0]
[0]']

 block_16_depthwise (DepthwiseC  (None, 5, 5, 960)   8640        ['block_16_expand_relu[0]
[0]']
 onv2D)


 block_16_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840        ['block_16_depthwise[0]
[0]']
 rmalization)


 block_16_depthwise_relu (ReLU)  (None, 5, 5, 960)   0           ['block_16_depthwise_BN
[0][0]']

 block_16_project (Conv2D)      (None, 5, 5, 320)    307200      ['block_16_depthwise_relu
[0][0]']

 block_16_project_BN (BatchNorm  (None, 5, 5, 320)   1280        ['block_16_project[0]
[0]']
 alization)
```

```
 Conv_1 (Conv2D)                (None, 5, 5, 1280)   409600      ['block_16_project_BN[0]
[0]']


 Conv_1_bn (BatchNormalization)  (None, 5, 5, 1280)  5120        ['Conv_1[0][0]']


 out_relu (ReLU)                (None, 5, 5, 1280)   0           ['Conv_1_bn[0][0]']


==========================================================================================
========
Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112
__________________________________________________________________________________________
________
```

In [ ]:

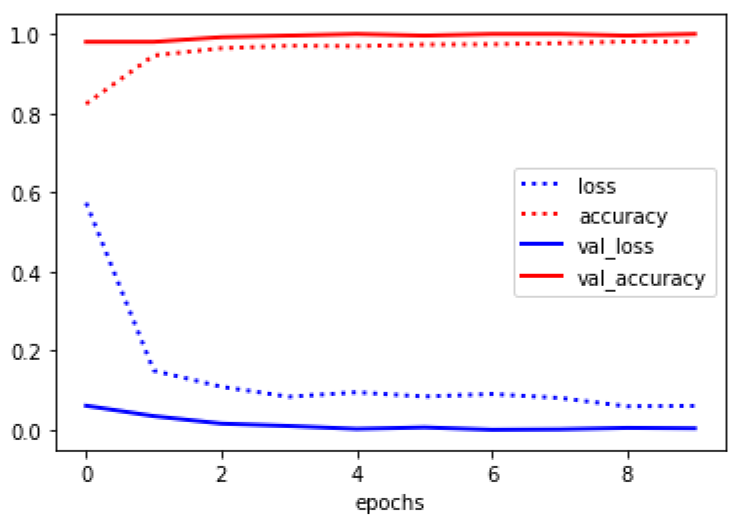
```
# Functions for creating transfer learning models

def transfer_rock_model():
  model = models.Sequential()
  conv_base.trainable = False
  model.add(conv_base)
  model.add(layers.GlobalAveragePooling2D())
  model.add(layers.Dropout(0.4))
  model.add(layers.Flatten())
  model.add(layers.Dense(256, activation='relu'))
  model.add(layers.Dropout(0.4))
  model.add(layers.Dense(3, activation='softmax'))
  model.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

  return model

def transfer_flower_model():
  model = models.Sequential()
  model.add(conv_base)
  model.add(layers.GlobalAveragePooling2D())
  model.add(layers.Flatten())
  model.add(layers.Dense(256, activation='relu'))
  model.add(layers.Dropout(0.5))
  model.add(layers.Dense(5, activation='softmax'))
  conv_base.trainable = True
  set_trainable = False
  for layer in conv_base.layers:
      if layer.name == 'block_16_expand':
          set_trainable = True
      if set_trainable:
          layer.trainable = True
      else:
          layer.trainable = False
  model.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model
```

In [ ]:

```
# Train rock model with fine tuning
```

```
run_evaluation("transfer_model_1_rock", transfer_rock_model, augment_data_rock(), base_di
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 15s 193ms/step - loss: 0.5737 - accuracy: 0.8236
 - val_loss: 0.0615 - val_accuracy: 0.9802
Epoch 2/10
60/60 [==============================] - 11s 179ms/step - loss: 0.1502 - accuracy: 0.9458
 - val_loss: 0.0352 - val_accuracy: 0.9802
Epoch 3/10
60/60 [==============================] - 11s 175ms/step - loss: 0.1094 - accuracy: 0.9641
 - val_loss: 0.0162 - val_accuracy: 0.9921
Epoch 4/10
60/60 [==============================] - 11s 178ms/step - loss: 0.0841 - accuracy: 0.9703
 - val_loss: 0.0103 - val_accuracy: 0.9960
Epoch 5/10
60/60 [==============================] - 11s 175ms/step - loss: 0.0952 - accuracy: 0.9692
 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 6/10
60/60 [==============================] - 11s 177ms/step - loss: 0.0847 - accuracy: 0.9734
 - val_loss: 0.0068 - val_accuracy: 0.9960
Epoch 7/10
60/60 [==============================] - 11s 176ms/step - loss: 0.0909 - accuracy: 0.9739
 - val_loss: 6.9139e-04 - val_accuracy: 1.0000
Epoch 8/10
60/60 [==============================] - 11s 176ms/step - loss: 0.0810 - accuracy: 0.9770
 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 9/10
60/60 [==============================] - 11s 180ms/step - loss: 0.0599 - accuracy: 0.9807
 - val_loss: 0.0055 - val_accuracy: 0.9960
Epoch 10/10
60/60 [==============================] - 11s 177ms/step - loss: 0.0610 - accuracy: 0.9802
 - val_loss: 0.0043 - val_accuracy: 1.0000
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 100.00%
```



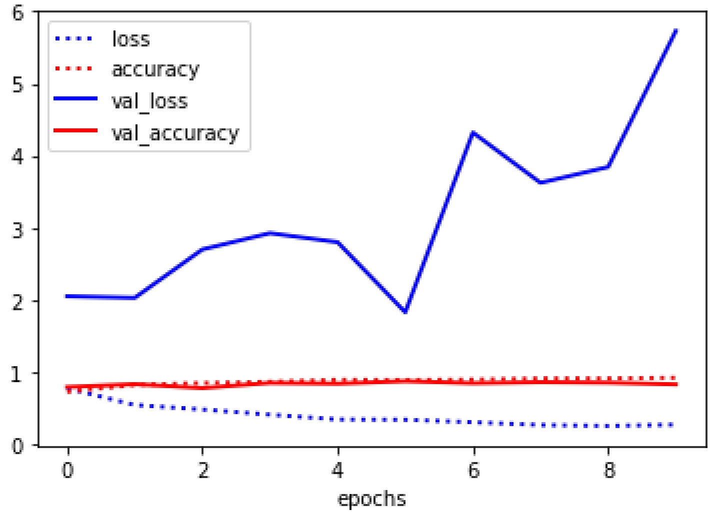

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984   
 ional)                                                          
                                                                 
 global_average_pooling2d (G  (None, 1280)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout_22 (Dropout)        (None, 1280)              0         
```

```
 flatten_5 (Flatten)         (None, 1280)              0

 dense_10 (Dense)            (None, 256)               327936

 dropout_23 (Dropout)        (None, 256)               0

 dense_11 (Dense)            (None, 3)                 771

=================================================================
Total params: 2,586,691
Trainable params: 328,707
Non-trainable params: 2,257,984
_________________________________________________________________
None
```

In [ ]:

```
# Train flower model with fine-tuning
run_evaluation("transfer_model_final_flower", transfer_flower_model, augment_data_flower()
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 16s 209ms/step - loss: 0.7777 - accuracy: 0.7267
 - val_loss: 2.0505 - val_accuracy: 0.7925
Epoch 2/10
60/60 [==============================] - 11s 190ms/step - loss: 0.5444 - accuracy: 0.8221
 - val_loss: 2.0294 - val_accuracy: 0.8333
Epoch 3/10
60/60 [==============================] - 12s 205ms/step - loss: 0.4821 - accuracy: 0.8489
 - val_loss: 2.7001 - val_accuracy: 0.7789
Epoch 4/10
60/60 [==============================] - 11s 182ms/step - loss: 0.4107 - accuracy: 0.8657
 - val_loss: 2.9253 - val_accuracy: 0.8503
Epoch 5/10
60/60 [==============================] - 11s 180ms/step - loss: 0.3440 - accuracy: 0.8935
 - val_loss: 2.8003 - val_accuracy: 0.8401
Epoch 6/10
60/60 [==============================] - 11s 181ms/step - loss: 0.3402 - accuracy: 0.8903
 - val_loss: 1.8295 - val_accuracy: 0.8741
Epoch 7/10
60/60 [==============================] - 11s 181ms/step - loss: 0.3054 - accuracy: 0.8961
 - val_loss: 4.3196 - val_accuracy: 0.8469
Epoch 8/10
60/60 [==============================] - 11s 180ms/step - loss: 0.2672 - accuracy: 0.9129
 - val_loss: 3.6231 - val_accuracy: 0.8605
Epoch 9/10
60/60 [==============================] - 11s 179ms/step - loss: 0.2544 - accuracy: 0.9129
 - val_loss: 3.8426 - val_accuracy: 0.8537
Epoch 10/10
60/60 [==============================] - 11s 177ms/step - loss: 0.2718 - accuracy: 0.9219
 - val_loss: 5.7272 - val_accuracy: 0.8299
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 87.41%
```




```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

 flatten_6 (Flatten)         (None, 1280)              0

 dense_12 (Dense)            (None, 256)               327936

 dropout_24 (Dropout)        (None, 256)               0

 dense_13 (Dense)            (None, 5)                 1285

=================================================================
Total params: 2,587,205
Trainable params: 1,215,301
Non-trainable params: 1,371,904
_________________________________________________________________
None
```

Checking the test accuracy for both models

In [ ]:

```
from sklearn.metrics import accuracy_score
base_dir = '/content/drive/My Drive/assignment'
name_rock = 'transfer_model_1_rock'
name_flower = 'transfer_model_final_flower'
model_rock = load_model_from_file(base_dir, name_rock, extension='.h5')
model_flower = load_model_from_file(base_dir, name_flower, extension='.h5')

yr_pred = model_rock.predict(Xr_test)
yf_pred = model_flower.predict(Xf_test)
score_r = accuracy_score(np.argmax(yr_test, axis=1), np.argmax(yr_pred, axis=1))
score_f = accuracy_score(np.argmax(yf_test, axis=1), np.argmax(yf_pred, axis=1))
print("Test accuracy rock paper scissors model: {:.2f}".format(score_r))
print("Test accuracy flower model: {:.2f}".format(score_f))
```

```
12/12 [==============================] - 1s 28ms/step
23/23 [==============================] - 1s 22ms/step
Test accuracy rock paper scissors model: 1.00
Test accuracy flower model: 0.85
```

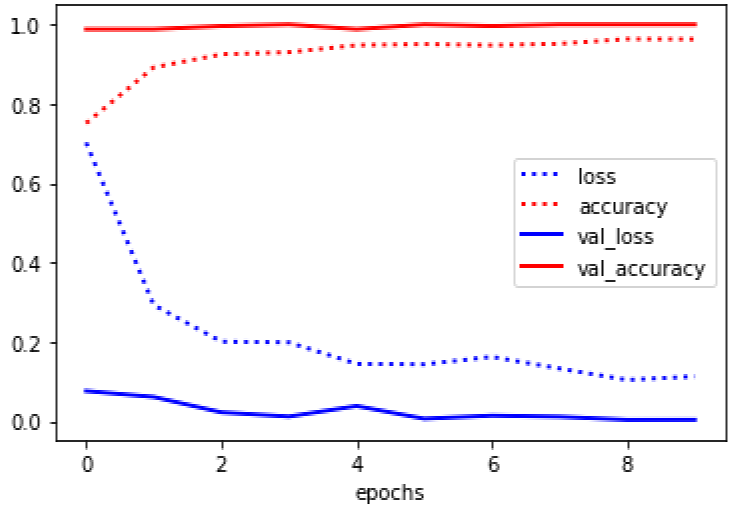
Trying other convolutional bases

In [ ]:
```python
# Trying a different convolutional base

# Import DenseNet121 convolutional base
from tensorflow.keras.applications.densenet import DenseNet121
conv_base = DenseNet121(input_shape=(IMG_SIZE, IMG_SIZE, 3),
                         include_top=False,
                         weights="imagenet",
                        )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densene
t/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 [==============================] - 3s 0us/step
```

In [ ]:
```python
run_evaluation("transfer_model_final2_rock", transfer_rock_model, augment_data_rock(), bas
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 24s 269ms/step - loss: 0.7022 - accuracy: 0.7510
 - val_loss: 0.0760 - val_accuracy: 0.9881
Epoch 2/10
60/60 [==============================] - 15s 244ms/step - loss: 0.2927 - accuracy: 0.8920
 - val_loss: 0.0610 - val_accuracy: 0.9881
Epoch 3/10
60/60 [==============================] - 16s 263ms/step - loss: 0.2005 - accuracy: 0.9248
 - val_loss: 0.0222 - val_accuracy: 0.9960
Epoch 4/10
60/60 [==============================] - 13s 214ms/step - loss: 0.1985 - accuracy: 0.9301
 - val_loss: 0.0116 - val_accuracy: 1.0000
Epoch 5/10
60/60 [==============================] - 17s 282ms/step - loss: 0.1445 - accuracy: 0.9478
 - val_loss: 0.0382 - val_accuracy: 0.9881
Epoch 6/10
60/60 [==============================] - 16s 269ms/step - loss: 0.1432 - accuracy: 0.9504
 - val_loss: 0.0061 - val_accuracy: 1.0000
Epoch 7/10
60/60 [==============================] - 15s 251ms/step - loss: 0.1623 - accuracy: 0.9478
 - val_loss: 0.0140 - val_accuracy: 0.9960
Epoch 8/10
60/60 [==============================] - 13s 216ms/step - loss: 0.1325 - accuracy: 0.9515
 - val_loss: 0.0111 - val_accuracy: 1.0000
Epoch 9/10
60/60 [==============================] - 12s 195ms/step - loss: 0.1041 - accuracy: 0.9635
 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 10/10
60/60 [==============================] - 12s 206ms/step - loss: 0.1127 - accuracy: 0.9629
 - val_loss: 0.0035 - val_accuracy: 1.0000
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 100.00%
```



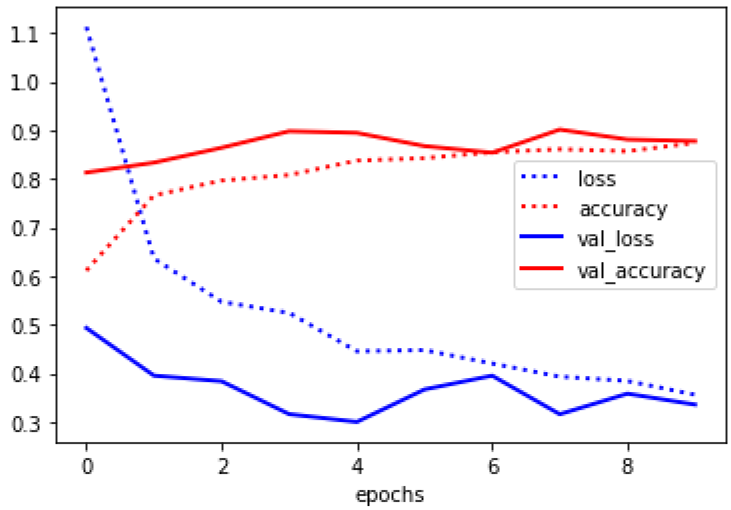

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 5, 5, 1024)        7037504

 global_average_pooling2d_2   (None, 1024)             0
 (GlobalAveragePooling2D)

 dropout_25 (Dropout)        (None, 1024)              0

 flatten_7 (Flatten)         (None, 1024)              0

 dense_14 (Dense)            (None, 256)               262400

 dropout_26 (Dropout)        (None, 256)               0

 dense_15 (Dense)            (None, 3)                 771

=================================================================
Total params: 7,300,675
Trainable params: 263,171
Non-trainable params: 7,037,504
_________________________________________________________________
None
```

In [ ]:

```
# DenseNet121 with the flower dataset
run_evaluation("transfer_model_final2_flower", transfer_flower_model, augment_data_flower
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 21s 259ms/step - loss: 1.1125 - accuracy: 0.6112
 - val_loss: 0.4932 - val_accuracy: 0.8129
Epoch 2/10
60/60 [==============================] - 12s 197ms/step - loss: 0.6359 - accuracy: 0.7660
 - val_loss: 0.3950 - val_accuracy: 0.8333
Epoch 3/10
60/60 [==============================] - 19s 310ms/step - loss: 0.5469 - accuracy: 0.7964
 - val_loss: 0.3835 - val_accuracy: 0.8639
Epoch 4/10
60/60 [==============================] - 14s 232ms/step - loss: 0.5240 - accuracy: 0.8080
 - val_loss: 0.3157 - val_accuracy: 0.8980
Epoch 5/10
60/60 [==============================] - 15s 240ms/step - loss: 0.4452 - accuracy: 0.8375
 - val_loss: 0.2998 - val_accuracy: 0.8946
Epoch 6/10
60/60 [==============================] - 15s 240ms/step - loss: 0.4476 - accuracy: 0.8426
```

```
 - val_loss: 0.3666 - val_accuracy: 0.8673
Epoch 7/10
60/60 [==============================] - 15s 241ms/step - loss: 0.4201 - accuracy: 0.8547
 - val_loss: 0.3950 - val_accuracy: 0.8537
Epoch 8/10
60/60 [==============================] - 13s 214ms/step - loss: 0.3930 - accuracy: 0.8610
 - val_loss: 0.3157 - val_accuracy: 0.9014
Epoch 9/10
60/60 [==============================] - 19s 309ms/step - loss: 0.3841 - accuracy: 0.8568
 - val_loss: 0.3580 - val_accuracy: 0.8810
Epoch 10/10
60/60 [==============================] - 16s 263ms/step - loss: 0.3557 - accuracy: 0.8740
 - val_loss: 0.3354 - val_accuracy: 0.8776
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 90.14%
```




```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 5, 5, 1024)        7037504

 global_average_pooling2d_3   (None, 1024)             0
 (GlobalAveragePooling2D)

 flatten_8 (Flatten)         (None, 1024)              0

 dense_16 (Dense)            (None, 256)               262400

 dropout_27 (Dropout)        (None, 256)               0

 dense_17 (Dense)            (None, 5)                 1285

=================================================================
Total params: 7,301,189
Trainable params: 263,685
Non-trainable params: 7,037,504
_________________________________________________________________
None
```

In [ ]:

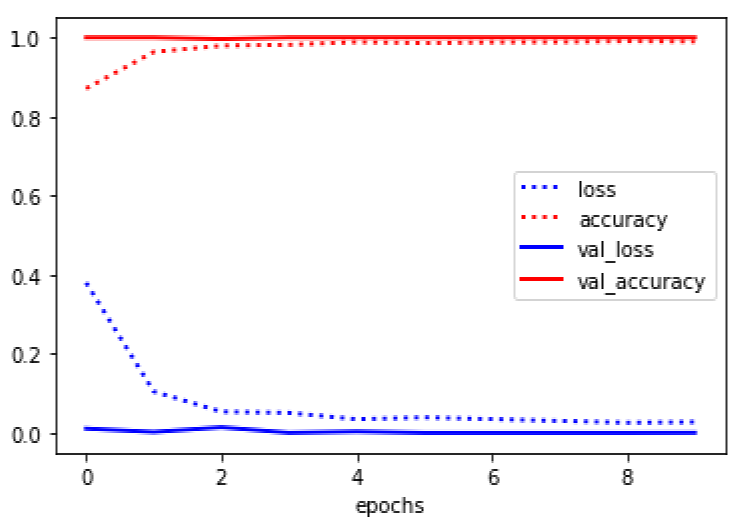
```
# Trying another convolutional base

# Import MobileNet convolutional base
from tensorflow.keras.applications import MobileNet
conv_base = MobileNet(input_shape=(IMG_SIZE, IMG_SIZE, 3),
                         include_top=False,
                         weights="imagenet",
                        )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilen
et/mobilenet_1_0_160_tf_no_top.h5
17225924/17225924 [==============================] - 3s 0us/step
```

In [ ]:

```python
# MobileNet with the rock paper scissors dataset
run_evaluation("transfer_model_final3_rock", transfer_rock_model, augment_data_rock(), bas
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 15s 201ms/step - loss: 0.3800 - accuracy: 0.8708
 - val_loss: 0.0109 - val_accuracy: 1.0000
Epoch 2/10
60/60 [==============================] - 10s 172ms/step - loss: 0.1046 - accuracy: 0.9635
 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 3/10
60/60 [==============================] - 10s 173ms/step - loss: 0.0535 - accuracy: 0.9791
 - val_loss: 0.0141 - val_accuracy: 0.9960
Epoch 4/10
60/60 [==============================] - 14s 232ms/step - loss: 0.0507 - accuracy: 0.9817
 - val_loss: 2.0709e-04 - val_accuracy: 1.0000
Epoch 5/10
60/60 [==============================] - 10s 172ms/step - loss: 0.0345 - accuracy: 0.9885
 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 6/10
60/60 [==============================] - 10s 170ms/step - loss: 0.0390 - accuracy: 0.9859
 - val_loss: 2.2382e-05 - val_accuracy: 1.0000
Epoch 7/10
60/60 [==============================] - 10s 173ms/step - loss: 0.0347 - accuracy: 0.9880
 - val_loss: 1.7130e-05 - val_accuracy: 1.0000
Epoch 8/10
60/60 [==============================] - 10s 170ms/step - loss: 0.0302 - accuracy: 0.9885
 - val_loss: 3.2310e-05 - val_accuracy: 1.0000
Epoch 9/10
60/60 [==============================] - 12s 193ms/step - loss: 0.0259 - accuracy: 0.9906
 - val_loss: 1.4249e-05 - val_accuracy: 1.0000
Epoch 10/10
60/60 [==============================] - 16s 264ms/step - loss: 0.0275 - accuracy: 0.9896
 - val_loss: 4.1770e-04 - val_accuracy: 1.0000
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 100.00%
```



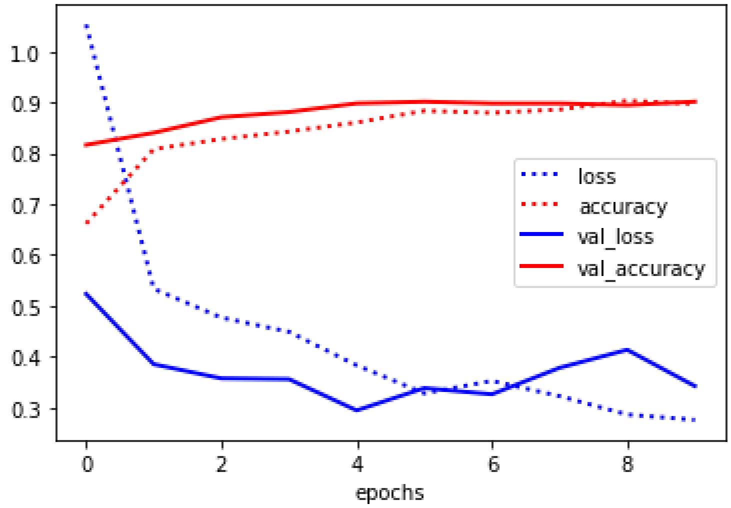

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_160 (Functio  (None, 5, 5, 1024)       3228864
 nal)
```

```
 global_average_pooling2d_4   (None, 1024)             0
 (GlobalAveragePooling2D)

 dropout_28 (Dropout)        (None, 1024)              0

 flatten_9 (Flatten)         (None, 1024)              0

 dense_18 (Dense)            (None, 256)               262400

 dropout_29 (Dropout)        (None, 256)               0

 dense_19 (Dense)            (None, 3)                 771

=================================================================
Total params: 3,492,035
Trainable params: 263,171
Non-trainable params: 3,228,864
_________________________________________________________________
None
```

In [ ]:

```
# MobileNet with flower dataset
run_evaluation("transfer_model_3_flower", transfer_flower_model, augment_data_flower(), ba
               train=True, generator=True, epochs=10, batch_size=None)
```

```
Training the model
Epoch 1/10
60/60 [==============================] - 17s 255ms/step - loss: 1.0531 - accuracy: 0.6616
 - val_loss: 0.5234 - val_accuracy: 0.8163
Epoch 2/10
60/60 [==============================] - 11s 188ms/step - loss: 0.5335 - accuracy: 0.8080
 - val_loss: 0.3851 - val_accuracy: 0.8401
Epoch 3/10
60/60 [==============================] - 10s 169ms/step - loss: 0.4768 - accuracy: 0.8279
 - val_loss: 0.3573 - val_accuracy: 0.8707
Epoch 4/10
60/60 [==============================] - 10s 171ms/step - loss: 0.4489 - accuracy: 0.8426
 - val_loss: 0.3558 - val_accuracy: 0.8810
Epoch 5/10
60/60 [==============================] - 10s 171ms/step - loss: 0.3829 - accuracy: 0.8604
 - val_loss: 0.2944 - val_accuracy: 0.8980
Epoch 6/10
60/60 [==============================] - 10s 174ms/step - loss: 0.3280 - accuracy: 0.8833
 - val_loss: 0.3382 - val_accuracy: 0.9014
Epoch 7/10
60/60 [==============================] - 10s 171ms/step - loss: 0.3524 - accuracy: 0.8793
 - val_loss: 0.3261 - val_accuracy: 0.8980
Epoch 8/10
60/60 [==============================] - 11s 184ms/step - loss: 0.3225 - accuracy: 0.8859
 - val_loss: 0.3781 - val_accuracy: 0.8980
Epoch 9/10
60/60 [==============================] - 10s 172ms/step - loss: 0.2862 - accuracy: 0.9031
 - val_loss: 0.4135 - val_accuracy: 0.8946
Epoch 10/10
60/60 [==============================] - 12s 192ms/step - loss: 0.2758 - accuracy: 0.8972
 - val_loss: 0.3423 - val_accuracy: 0.9014
Saving to file
Model stored in /content/drive/My Drive/assignment
Max val score: 90.14%
```




```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 mobilenet_1.00_160 (Functio  (None, 5, 5, 1024)       3228864   
 nal)                                                            
                                                                 
 global_average_pooling2d_5   (None, 1024)             0         
 (GlobalAveragePooling2D)                                        
                                                                 
 flatten_10 (Flatten)        (None, 1024)              0         
                                                                 
 dense_20 (Dense)            (None, 256)               262400    
                                                                 
 dropout_30 (Dropout)        (None, 256)               0         
                                                                 
 dense_21 (Dense)            (None, 5)                 1285      
                                                                 
=================================================================
Total params: 3,492,549
Trainable params: 263,685
Non-trainable params: 3,228,864
_________________________________________________________________
None
```

## 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.

## For the rock dataset

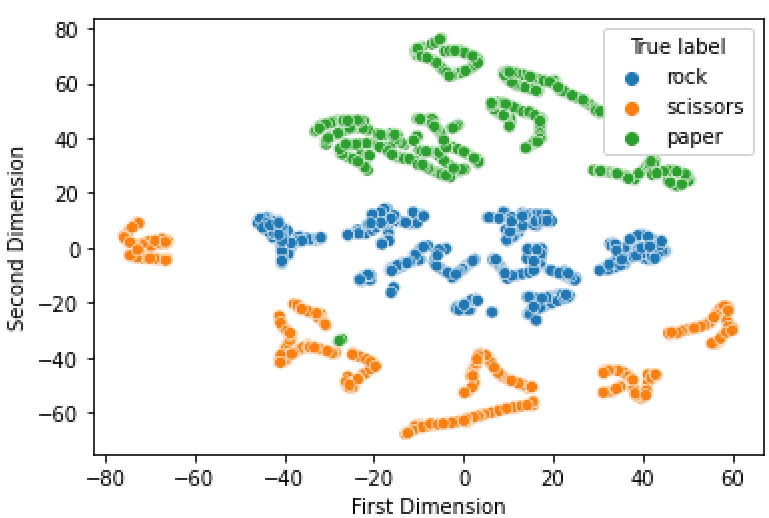
In [ ]:
```
#Extract embeddings of training images

#loading model
base_dir = '/content/drive/My Drive/assignment'
name = "transfer_model_final_rock"
model = load_model_from_file(base_dir, name, extension='.h5')
```

In [ ]:
```
from sklearn.manifold import TSNE
embeddings = models.Model(inputs = model.inputs, outputs = [model.get_layer('global_averag
embed_predictions = embeddings.predict(Xr_train)

reduced_embed_predictions = TSNE(n_components=2).fit_transform(embed_predictions)
```

```
71/71 [==============================] - 3s 18ms/step
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The
default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The
default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
```

In [ ]:
```
import seaborn
df_for_plot = pd.DataFrame(columns=['First Dimension', 'Second Dimension', 'True label'])
for i in range(0, len(reduced_embed_predictions)):
  df_for_plot = df_for_plot.append({'First Dimension': reduced_embed_predictions[i][0],
                                    'Second Dimension': reduced_embed_predictions[i][1],
                                    'True label': class_names_r[np.argmax(yr_train[i])]},

seaborn.scatterplot(data = df_for_plot, x='First Dimension', y='Second Dimension', hue='Tr
# seaborn.set(rc = {'figure.figsize':(18,5)})
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bd159ee10>
```




## For the flower dataset

In [ ]:
```
base_dir = '/content/drive/My Drive/assignment'
name = "transfer_model_3_flower"
model = load_model_from_file(base_dir, name, extension='.h5')
```

In [ ]:
```
from sklearn.manifold import TSNE
embeddings = models.Model(inputs = model.inputs, outputs = [model.get_layer('global_averag
embed_predictions = embeddings.predict(Xf_train)
```

```
_
reduced_embed_predictions = TSNE(n_components=2).fit_transform(embed_predictions)
```

```
83/83 [==============================] - 2s 15ms/step
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
```

In [ ]:

```
import seaborn
df_for_plot = pd.DataFrame(columns=['First Dimension', 'Second Dimension', 'True label'])
for i in range(0, len(reduced_embed_predictions)):
  df_for_plot = df_for_plot.append({'First Dimension': reduced_embed_predictions[i][0],
                                   'Second Dimension': reduced_embed_predictions[i][1],
                                   'True label': class_names_f[np.argmax(yf_train[i])]},

seaborn.scatterplot(data = df_for_plot, x='First Dimension', y='Second Dimension', hue='Tr
seaborn.set(rc = {'figure.figsize':(25,8)})
```


## 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?

We tried to finish bonus question, but due to time constraints and unforseen difficulties, we couldn't get the plots to work

In [ ]:

```
activations = models.Model(inputs = model.inputs, outputs = [layer.output for layer in moc
activations_predictions = activations.predict(Xr_train)
_
reduced_activations_predictions = TSNE(n_components=2).fit_transform(activations_predictic
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-78-f9a6a9553374> in <module>
----> 1 activations = models.Model(inputs = model.inputs, outputs = [layer.output for laye
r in model.layers[:15]])
      2 activations_predictions = activations.predict(Xr_train)
      3 _
      4 reduced_activations_predictions = TSNE(n_components=2).fit_transform(activations_p
redictions)

/usr/local/lib/python3.7/dist-packages/tensorflow/python/training/tracking/base.py in _met
hod_wrapper(self, *args, **kwargs)
```

```
    585     self._self_setattr_tracking = False  # pylint: disable=protected-access
    586     try:
--> 587       result = method(self, *args, **kwargs)
    588     finally:
    589       self._self_setattr_tracking = previous_value  # pylint: disable=protected-ac
cess

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py in __init__(self, input
s, outputs, name, trainable, **kwargs)
    146                  for t in tf.nest.flatten(inputs)]):
    147         inputs, outputs = functional_utils.clone_graph_nodes(inputs, outputs)
--> 148     self._init_graph_network(inputs, outputs)
    149 
    150   @tf.__internal__.tracking.no_automatic_dependency_tracking

/usr/local/lib/python3.7/dist-packages/tensorflow/python/training/tracking/base.py in _met
hod_wrapper(self, *args, **kwargs)
    585     self._self_setattr_tracking = False  # pylint: disable=protected-access
    586     try:
--> 587       result = method(self, *args, **kwargs)
    588     finally:
    589       self._self_setattr_tracking = previous_value  # pylint: disable=protected-ac
cess

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py in _init_graph_network(s
elf, inputs, outputs)
    231     # Keep track of the network's nodes and layers.
    232     nodes, nodes_by_depth, layers, _ = _map_graph_network(
--> 233         self.inputs, self.outputs)
    234     self._network_nodes = nodes
    235     self._nodes_by_depth = nodes_by_depth

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py in _map_graph_network(in
puts, outputs)
    997           if id(x) not in computable_tensors:
    998             raise ValueError(
--> 999                 f'Graph disconnected: cannot obtain value for tensor {x} '
   1000                 f'at layer "{layer.name}". The following previous layers '
   1001                 f'were accessed without issue: {layers_with_complete_input}')

ValueError: Graph disconnected: cannot obtain value for tensor KerasTensor(type_spec=Tenso
rSpec(shape=(None, 160, 160, 3), dtype=tf.float32, name='input_4'), name='input_4', descri
ption="created by layer 'input_4'") at layer "conv1". The following previous layers were a
ccessed without issue: []
```

In [ ]:

```python
from tensorflow.keras import models

img_tensor = Xr_test[4]
img_tensor = np.expand_dims(img_tensor, axis=0)

# Extracts the outputs of the top 8 layers:
layer_outputs = [layer.output for layer in model.layers[:15]]
print(layer_outputs)
# Creates a model that will return these outputs, given the model input:
# activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

# This will return a list of 5 Numpy arrays:
# one array per layer activation
activations = activation_model.predict(img_tensor)
```

```
[<KerasTensor: shape=(None, 5, 5, 1024) dtype=float32 (created by layer 'conv_pw_13_relu')
>, <KerasTensor: shape=(None, 1024) dtype=float32 (created by layer 'global_average_poolin
g2d_5')>, <KerasTensor: shape=(None, 1024) dtype=float32 (created by layer 'flatten_9')>,
<KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'dense_18')>, <KerasTenso
r: shape=(None, 256) dtype=float32 (created by layer 'dropout_26')>, <KerasTensor: shape=
```

```
(None, 5) dtype=float32 (created by layer 'dense_19')>]
1/1 [==============================] - 0s 53ms/step
```

In [ ]:

```

```

In [ ]:

```
plt.rcParams['figure.dpi'] = 120
first_layer_activation = reduced_embed_predictions

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.imshow(img_tensor[0])
ax2.matshow(first_layer_activation)
ax1.set_xticks([])
ax1.set_yticks([])
ax2.set_xticks([])
ax2.set_yticks([])
ax1.set_xlabel('Input image')
ax2.set_xlabel('Activation of filter 2');
```


## Have fun!