Deep Learning Project: *Rice Image Dataset*

To develop the project we use the dataset available on Kaggle at the following link: https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset (https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset). The dataset is called *Rice Image Dataset*.

- It contains 75K images including 15K images for each rice variety that are:
- [Arborio, Basmati, Ipsala, Jasmine and Karacadag]

We decide to use the dataset to solve a **classification problem**. We want to find a performing deep learning model to correctly classify 5 types of rice.

We mainly use two approaches

- **Convolutional Neaural Network from scratch (CNN)**
- **Transfer learning and Fine Tuning and feature extraction**

In the first case we design a new architecture of a Convolutional Neural Network from scratch.

In the second case we use completely a Pre-trained Neural Network adding a Feed-Forward Neural Network to improve the performances.

Neural networks with the best performances are presented in this notebook.

Notebook 1 and ***Notebook 2*** contain all the attempts.

To finally choose the best model, we created a new dataset, called *test* on which we fitted the two models. The model with better classificatory goodness-of-fit is chosen as the best model.

1) Environment Set Up

This section is made to configure the environment, import useful libraries and set up the directories:

In [ ]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import pandas as pd
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from keras import layers
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout, GlobalAveragePooling2D
from keras.models import Sequential,  Model
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import resnet50, MobileNet,VGG16
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet50
from tensorflow.keras.applications.mobilenet import preprocess_input as preprocess_inputMN
from tensorflow.keras.applications.vgg16 import preprocess_input as preprocess_inputVG

```

In [ ]:

```
# Import main libraries

from time import time
from datetime import datetime
from google.colab import drive

import cv2
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import load_img
print('TensorFlow version:', tf.__version__)

import zipfile
from shutil import copyfile


import numpy as np
import pandas as pd
import random as python_random

import glob
import shutil
from random import seed
from random import random
import os
import os.path


from IPython.display import Javascript

```

```
TensorFlow version: 2.8.2
```

In [ ]:

```
# Mount GDrive
drive.mount('/content/gdrive/', force_remount = True)
```

```
<IPython.core.display.Javascript object>

Mounted at /content/gdrive/
```

2) Image Loading and Sampling

In our case, the data preparation phase for further analysis consists of two steps: **loading data onto the drive** and **sampling**.

1. Unzipping and loading the original dataset with 75K images in total, particularly 15K per class
2. Sampling the data in order to train the model faster

The computational time with which this operation is performed is 5 hours and 46 minutes.

In [ ]:
```
# t0 = time()

# # copyfile('rice.zip', 'rice.zip')

# zip = zipfile.ZipFile('rice.zip')
# zip.extractall()
# zip.close()

# print("File transfer completed in %0.3f seconds" % (time() - t0))
```

```
<IPython.core.display.Javascript object>
```

We now define a new function called *random_sampling()* that randomly samples the original dataset by extracting *n* images per class, where *n* is defined by us.

In this case we extract 2K images per class, for a total of 10K images. Recall that the target classes are equidistributed.

A *seed* is set for reproducibility.

In [ ]:

```
def random_sampling(index, src_dir=None, dst_dir=None):

  for riso in os.listdir(src_dir):

    dir = src_dir + str(riso) # Source dir for every rice type
    dst = dst_dir + str(riso) # Destination dir for every rice type

    if(os.path.exists(dst)):
      os.remove(dst)

    if not os.path.isdir(dst): #create folder for every rice type in not exist
      os.makedirs(dst)

    for i, jpgfile in enumerate(glob.iglob(os.path.join(dir, "*.jpg"))):
      if i in index:
        shutil.copy(jpgfile, dst)


```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

In [ ]:
```python
# Draw sample from the original dataset
from random import randint

# Index for Random Sampling
index = []
for _ in range(2000):
    value = randint(0, 15000)
    index.append(value)

# Define source dir and destination dir
src_dir = '/content/gdrive/MyDrive/FDL2022Project/Dati/Original/Rice_Image_Dataset/'
dst_dir = '/content/gdrive/MyDrive/FDL2022Project/Dati/Sample/Final/'

import random
random.seed(123)
random_sampling(index, src_dir=src_dir, dst_dir=dst_dir)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

3) EDA: Exploratory Data Analysis

Before entering the heart of the modeling phase, an exploratory analysis is performed to understand what the dataset looks like.

In [ ]:
```python
base_path = '/content/gdrive/MyDrive/FDL2022Project/Dati/Sample/Final/'
```

```
<IPython.core.display.Javascript object>
```

As previously mentioned the dataset consists of a set of images of rice grains classified into 5 types:

- Arborio
- Basamti
- Ipsala

- Jasmine
- Karacadag.

In [ ]:
```
rice_classes=os.listdir(base_path)
print(rice_classes)
```

```
<IPython.core.display.Javascript object>
```

```
['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

We print the 3 dimensions of the image *Arborio (10).jpg*: width, height and depth. We use the information of *Arborio (10).jpg* assuming that each image of the dataset has the same dimensions.

The images have dimensions 250x250x3 where 250 is the width, 250 is the height e 3 is the depth. Each image has 3 color channels, which means it is RGB type.

In [ ]:
```
image = cv2.imread(base_path + 'Arborio/Arborio (10).jpg')

print(type(image))

print('-------')

print('Width:', image.shape[0])
print('Height:', image.shape[1])
print('Channels:', image.shape[2])


```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

To see if the images were correctly imported and to explore the dataset, we decide to print the first image for each type of rice grain.

In [ ]:

```
plt.figure(0, figsize=(250,250))
cpt = 0

for riso in os.listdir(base_path):
    for i in range(1,2):
        cpt = cpt + 1
        sp=plt.subplot(7,5,cpt)
        sp.axis('Off')
        img_path = base_path + riso + "/" +os.listdir(base_path + riso+ "/")[i]
        img = load_img( img_path, target_size=(250,250))
        plt.imshow(img, cmap="gray")


plt.show()
```

```
<IPython.core.display.Javascript object>
```

From these outputs we notice how the rice grains in these 5 images differ from each other in characteristics such as length, width or shape.

In this example, the first image (type *Arborio*) is similar to the fifth (type *Karacadag*)

In the exploratory analysis, it is important to know the **distribution of the classes of the target variable** in the dataset that will be trained. It is possible that some classes are under-represented or over-represented, consequently the variable's classes may be unbalanced. New techniques should be used to deal with this problem.

In [ ]:

```
df_distribution_train = pd.DataFrame()
for riso in os.listdir(base_path):
    dir = base_path + str(riso)
    ln = len(os.listdir(dir))
    new = pd.DataFrame({'Expression':[riso], 'n_images':ln})
    df_distribution_train = pd.concat([df_distribution_train,new])
```

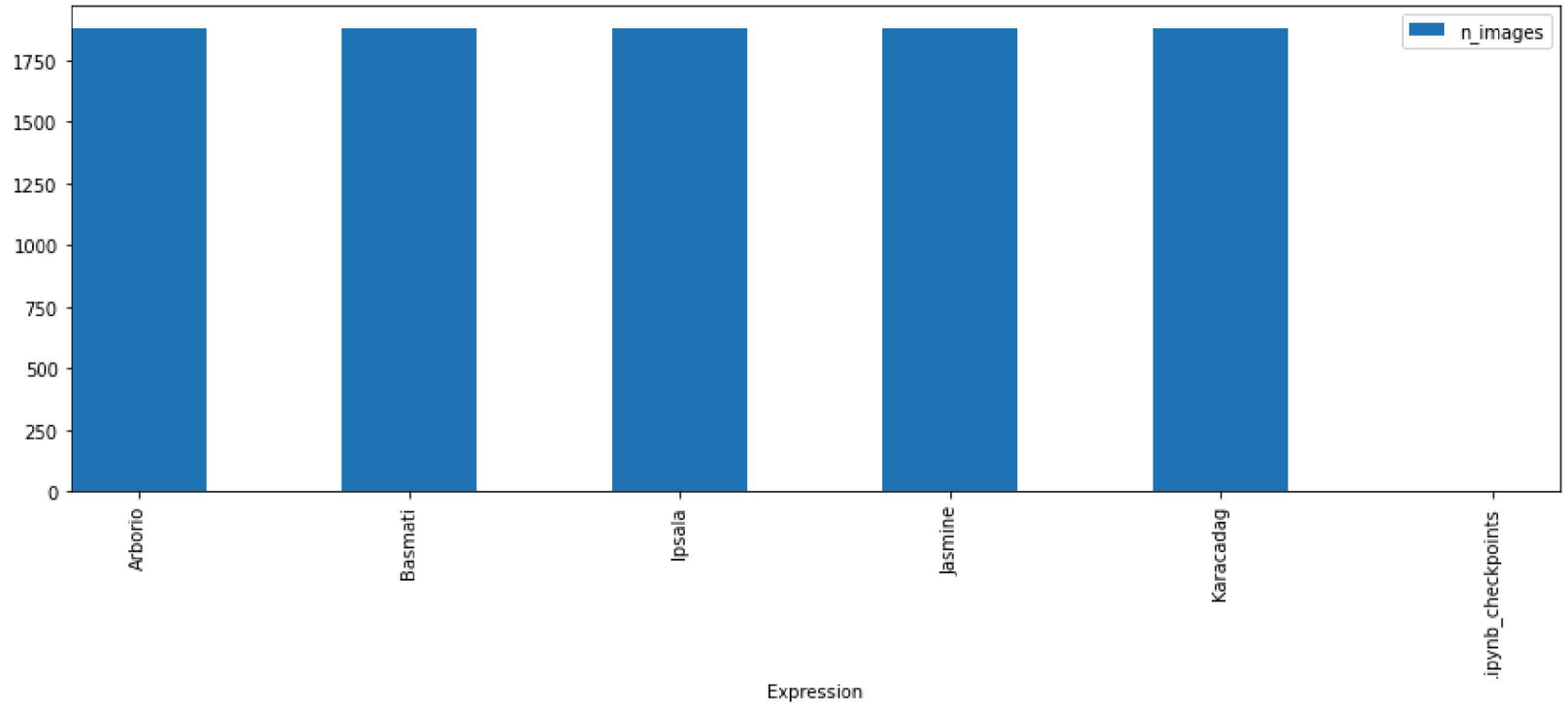
```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

In [ ]:
```
ax = df_distribution_train.plot.bar(x='Expression',y='n_images',figsize=(15, 5))
ax.autoscale(axis='x',tight=True);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



From this graph we can see how the target's classes are equidistributed. In fact, there are 2000 images for each type of rice grain. No further action will be taken in consideration.

##4) **Preprocessing** and **Splitting into Training and Validation**

4.1) Preprocessing and Splitting into Training and Validation for Best Model from scratch

Before proceeding to the actual analysis phase, it is necessary to split the whole dataset into two datasets: **training set** and **validation set**. 80% of the data is contained in the training while 20% of the data is contained in the validation.

On the training dataset, the data are standardized (*rescale=1.0/255.0*). In addition, to make the classificatory model as generalizable as possible on new data, the technique of **data augmentation** is used. Data augmentation creates a more complete and consistent set of data. We applied different techniques of data augmentation, in particular data warping techniques such as: *width shift*, *height shift*, *horizontal flip*, *vertical flip* an *rotation*. All this approaches trasform the datas preserving the original labels.

On the validation dataset, the data are only standardized (*rescale=1.0/255.0*).

Finally, two **data loaders** are defined.

The batch size is set to 32. The images are risized from 250x250 to 224x224 dimension.

In [ ]:
```
from keras.preprocessing.image import ImageDataGenerator
```

```
<IPython.core.display.Javascript object>
```

In [ ]:
```
# Training dataset: standardization + data augmentation

train_datagen = ImageDataGenerator(rescale=1.0/255.0,
                                   width_shift_range = 0.1,
                                   height_shift_range = 0.1,
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   rotation_range = 20,
                                   validation_split=0.2)
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

```
# Validation dataset: standardization

validation_datagen = ImageDataGenerator(rescale=1/255,
                                        validation_split=0.2)
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

```
# Training data loader

train_generator = train_datagen.flow_from_directory(base_path,
                                                    target_size=(224,224),
                                                    color_mode='rgb',
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    subset='training',
                                                    shuffle=True,
                                                    seed=1)

```

```
<IPython.core.display.Javascript object>
```

```
Found 7505 images belonging to 5 classes.
```

In [ ]:

```
# Validation data loader

validation_generator = validation_datagen.flow_from_directory(base_path,
                                                              target_size=(224,224),
                                                              color_mode='rgb',
                                                              batch_size=32,
                                                              class_mode='categorical',
                                                              subset='validation',
                                                              shuffle=False,
                                                              seed=1)
```

```
<IPython.core.display.Javascript object>
```

```
Found 1875 images belonging to 5 classes.
```

In [ ]:
```
for image_batch, labels_batch in train_generator:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
<IPython.core.display.Javascript object>

(32, 224, 224, 3)
(32, 5)
```

###4.2) **Preprocessing** and **Splitting into Training and Validation** for **Best Model with Transfer Learning**

As done in the previous section, a training dataset and a validation dataset are defined, with their respective data loaders. Compared with the previous case, in addition to standardization and data augmentation, **preprocessing of the ResNet50** network is added.

In [ ]:
```
size=224
channels=3
batch_size = 64
num_classes = 5

# Training set

train_processing = ImageDataGenerator(preprocessing_function=preprocess_input_resnet50,
                                      validation_split=0.2,
                                      rotation_range = 25,
                                      width_shift_range = .2,
                                      height_shift_range = .2,
                                      horizontal_flip = True,
                                      zoom_range = .2)
```

In [ ]:

```
# Validation set

validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input_resnet50,
                                        validation_split=0.2)
```

In [ ]:

```
# Training data loader

train_generator_tf = train_processing.flow_from_directory (base_path,
                                                           target_size=(size, size),
                                                           batch_size=batch_size,
                                                           shuffle=True,
                                                           class_mode='categorical',
                                                           subset='training',
                                                           color_mode="rgb",
                                                           interpolation='nearest')
```

In [ ]:

```
# Validation data loader

validation_generator_tf = validation_datagen.flow_from_directory(base_path,
                                                                 target_size=(size,size),
                                                                 subset='validation',
                                                                 color_mode="rgb",
                                                                 batch_size=batch_size,
                                                                 class_mode='categorical',
                                                                 shuffle=False,
                                                                 interpolation='nearest')
```

In [ ]:
```
for image_batch, labels_batch in train_generator:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
<IPython.core.display.Javascript object>

(64, 224, 224, 3)
(64, 5)
```

5) *Best Model*

###5.1) **Best Model from scratch**

The best model is the **Model_v7** where the neural network is named as **net7** (see Notebook 1).

This neural network is chosen as the best mainly for two reasons.
First, it has an accuracy on validation of 99%: the *val_accuracy* at the 40th epoch is 0.99. This means that the neural network correctly classifies 99% of images it receives as input after 40 epochs.
Second, the accuracy remains constant from the 25th epoch onward to the 40th epoch, which suggests that the model is stable.

In this case the model is trained for 30 epochs.

The model's architecture is presented in the cell below.

Architecture Model from scratch

- Input Layer
- 2d Convolutional layer (3x3 filter, 32 hidden neurons, Ridge regularizer, He uniform inizializer)
- Relu activation
- Batch Normalization layer
- Max Pooling layer (3x3)
- 2d convolutional Layer (3x3, 64 hidden neurons, Ridge regularizer, He uniform inizializer)
- Relu activation
- Batch Normalization layer
- Max Pooling layer (3x3)

- 2d Convolutional layer (3x3 filter, 128 hidden neurons, Ridge regularizer, He uniform inizializer)
- Relu activation
- Batch Normalization layer
- Global Max Pooling layer
- Dense layer (Softmax Activation, ridge regularizer)

In [ ]:
```
num_classes = 5
```

<IPython.core.display.Javascript object>

In [ ]:
```
# Architecture definition

inputs = keras.Input((224, 224, 3))

x = inputs
x = keras.layers.Conv2D(32, 3, padding = 'same', kernel_regularizer = tf.keras.regularizers.l2(), kernel
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Activation('relu')(x)
x = keras.layers.MaxPooling2D(3, strides = 3, padding = 'same')(x)

x = keras.layers.Conv2D(64, 3, padding = 'same', kernel_regularizer = tf.keras.regularizers.l2(), kernel
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Activation('relu')(x)
x = keras.layers.MaxPooling2D(3, strides = 3, padding = 'same')(x)

x = keras.layers.Conv2D(128, 3, padding = 'same', kernel_regularizer = tf.keras.regularizers.l2(), kerne
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Activation('relu')(x)

x = keras.layers.GlobalMaxPooling2D()(x)

outputs = keras.layers.Dense(num_classes, activation = 'softmax', kernel_regularizer=tf.keras.regularize
net1 = keras.Model(inputs, outputs)
```

<IPython.core.display.Javascript object>

A summary and plot of the neural network architecture described above is shown in the cells below. In particular, we note that the total number of parameters to be estimated is 94 789 of which 94 341 are trainable while 448 are not.

In [ ]:

```
# Architecture summary

net1.summary()
```

<IPython.core.display.Javascript object>

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 32)      896

 batch_normalization (BatchN  (None, 224, 224, 32)     128
 ormalization)

 activation (Activation)     (None, 224, 224, 32)      0

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 batch_normalization_1 (Batc  (None, 75, 75, 64)       256
 hNormalization)

 activation_1 (Activation)   (None, 75, 75, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 25, 25, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 25, 25, 128)       73856

 batch_normalization_2 (Batc  (None, 25, 25, 128)      512
 hNormalization)

 activation_2 (Activation)   (None, 25, 25, 128)       0

 global_max_pooling2d (Globa  (None, 128)              0
 lMaxPooling2D)

 dense (Dense)               (None, 5)                 645

=================================================================
Total params: 94,789
Trainable params: 94,341
```

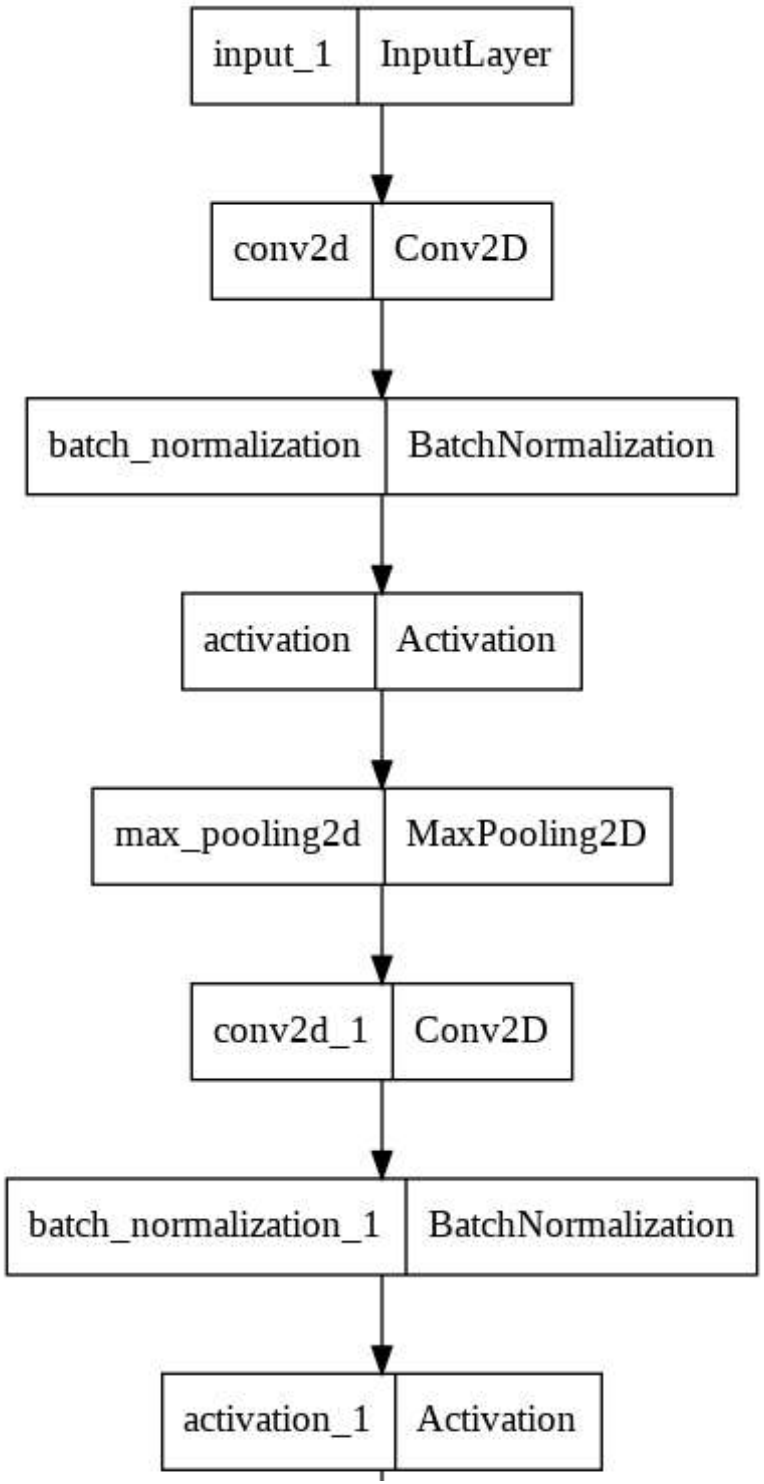
```
Non-trainable params: 448
_________________________________________________________________
```

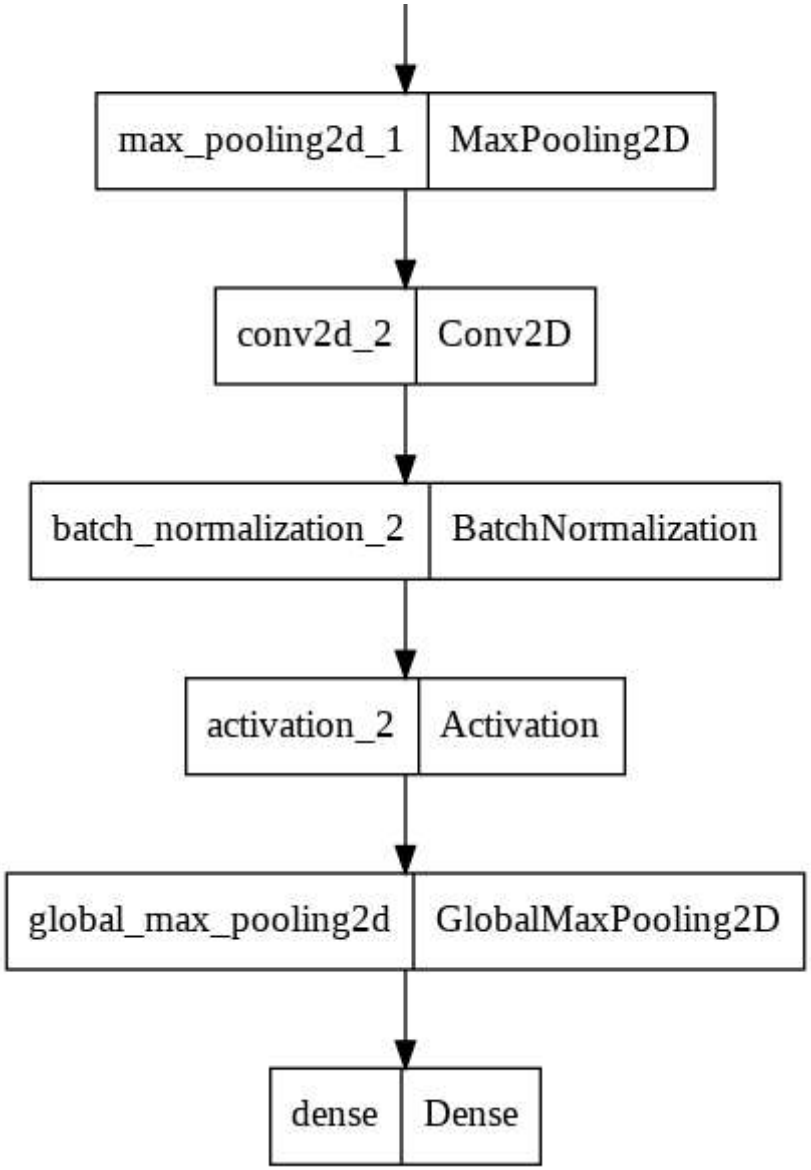
In [ ]:

```
# Architecture plot

keras.utils.plot_model(net1)
```

```
<IPython.core.display.Javascript object>
```

Out[19]:


input_1
InputLayer
conv2d
Conv2D
batch_normalization
BatchNormalization
activation
Activation
max_pooling2d
MaxPooling2D
conv2d_1
Conv2D
batch_normalization_1
BatchNormalization
activation_1
Activation



In the next step, the network is trained for the classification task. Specifically, **categorical crossentropy** is chosen as the loss function, **RMSprop** is chosen as the optimizer, and **accuracy** is chosen as the metric for evaluating classification goodness-of-fit.

In [ ]:

```
# Network Compilation

net1.compile(loss = keras.losses.categorical_crossentropy,
             optimizer = keras.optimizers.RMSprop(learning_rate=0.001),
             metrics =['accuracy'])
```

<IPython.core.display.Javascript object>

In [ ]:

```
# Reduce Learning Rate

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'val_loss',
                                                 mode='min',
                                                 patience=3,
                                                 verbose=1,
                                                 factor=0.5,
                                                 min_lr=0.000001)
```

<IPython.core.display.Javascript object>

In [ ]:

```
# Training

history1 = net1.fit(train_generator,
                    epochs = 30,
                    validation_data = validation_generator,
                    callbacks = [reduce_lr]);
```

<IPython.core.display.Javascript object>

```
Epoch 1/30
235/235 [==============================] - 2043s 9s/step - loss: 3.8973 - accuracy: 0.7472 - val_loss: 3.50
86 - val_accuracy: 0.2293 - lr: 0.0010
Epoch 2/30
235/235 [==============================] - 617s 3s/step - loss: 1.7010 - accuracy: 0.9037 - val_loss: 3.445
0 - val_accuracy: 0.2299 - lr: 0.0010
Epoch 3/30
235/235 [==============================] - 619s 3s/step - loss: 1.0733 - accuracy: 0.9332 - val_loss: 2.232
4 - val_accuracy: 0.4965 - lr: 0.0010
Epoch 4/30
235/235 [==============================] - 625s 3s/step - loss: 0.7973 - accuracy: 0.9402 - val_loss: 1.605
3 - val_accuracy: 0.5867 - lr: 0.0010
Epoch 5/30
235/235 [==============================] - 625s 3s/step - loss: 0.6522 - accuracy: 0.9446 - val_loss: 0.981
4 - val_accuracy: 0.8123 - lr: 0.0010
Epoch 6/30
235/235 [==============================] - 620s 3s/step - loss: 0.5706 - accuracy: 0.9459 - val_loss: 1.362
2 - val_accuracy: 0.6987 - lr: 0.0010
Epoch 7/30
235/235 [==============================] - 621s 3s/step - loss: 0.5191 - accuracy: 0.9458 - val_loss: 0.415
9 - val_accuracy: 0.9755 - lr: 0.0010
Epoch 8/30
235/235 [==============================] - 618s 3s/step - loss: 0.4746 - accuracy: 0.9548 - val_loss: 1.267
2 - val_accuracy: 0.6848 - lr: 0.0010
Epoch 9/30
235/235 [==============================] - 620s 3s/step - loss: 0.4279 - accuracy: 0.9587 - val_loss: 0.437
6 - val_accuracy: 0.9616 - lr: 0.0010
Epoch 10/30
235/235 [==============================] - ETA: 0s - loss: 0.4002 - accuracy: 0.9592
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
235/235 [==============================] - 617s 3s/step - loss: 0.4002 - accuracy: 0.9592 - val_loss: 1.422
8 - val_accuracy: 0.6992 - lr: 0.0010
Epoch 11/30
235/235 [==============================] - 618s 3s/step - loss: 0.3001 - accuracy: 0.9836 - val_loss: 0.326
1 - val_accuracy: 0.9733 - lr: 5.0000e-04
Epoch 12/30
235/235 [==============================] - 615s 3s/step - loss: 0.2689 - accuracy: 0.9864 - val_loss: 0.516
1 - val_accuracy: 0.8768 - lr: 5.0000e-04
Epoch 13/30
235/235 [==============================] - 617s 3s/step - loss: 0.2639 - accuracy: 0.9815 - val_loss: 0.527
9 - val_accuracy: 0.8853 - lr: 5.0000e-04
Epoch 14/30
235/235 [==============================] - 618s 3s/step - loss: 0.2567 - accuracy: 0.9800 - val_loss: 0.276
```

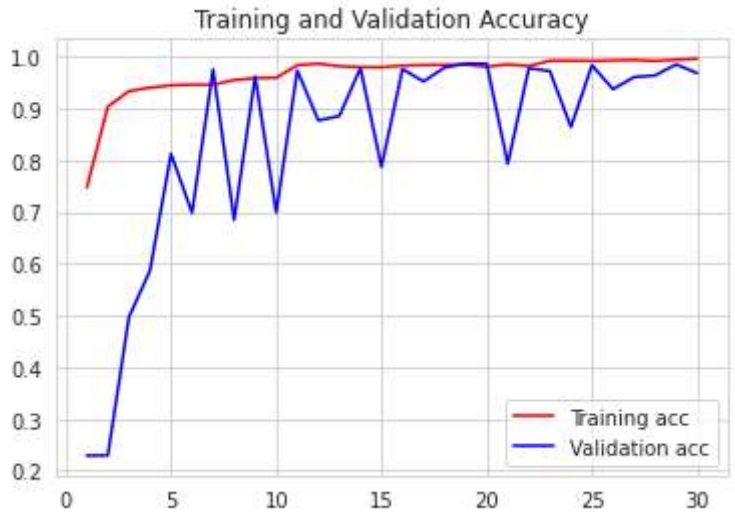
```
4 - val_accuracy: 0.9781 - lr: 5.0000e-04
Epoch 15/30
235/235 [==============================] - 617s 3s/step - loss: 0.2467 - accuracy: 0.9796 - val_loss: 0.674
4 - val_accuracy: 0.7872 - lr: 5.0000e-04
Epoch 16/30
235/235 [==============================] - 622s 3s/step - loss: 0.2385 - accuracy: 0.9825 - val_loss: 0.244
5 - val_accuracy: 0.9760 - lr: 5.0000e-04
Epoch 17/30
235/235 [==============================] - 621s 3s/step - loss: 0.2285 - accuracy: 0.9837 - val_loss: 0.326
5 - val_accuracy: 0.9520 - lr: 5.0000e-04
Epoch 18/30
235/235 [==============================] - 618s 3s/step - loss: 0.2251 - accuracy: 0.9841 - val_loss: 0.233
3 - val_accuracy: 0.9792 - lr: 5.0000e-04
Epoch 19/30
235/235 [==============================] - 621s 3s/step - loss: 0.2194 - accuracy: 0.9845 - val_loss: 0.195
5 - val_accuracy: 0.9867 - lr: 5.0000e-04
Epoch 20/30
235/235 [==============================] - 618s 3s/step - loss: 0.2168 - accuracy: 0.9805 - val_loss: 0.210
4 - val_accuracy: 0.9861 - lr: 5.0000e-04
Epoch 21/30
235/235 [==============================] - 620s 3s/step - loss: 0.2045 - accuracy: 0.9849 - val_loss: 0.801
3 - val_accuracy: 0.7931 - lr: 5.0000e-04
Epoch 22/30
235/235 [==============================] - ETA: 0s - loss: 0.2070 - accuracy: 0.9817
Epoch 22: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
235/235 [==============================] - 617s 3s/step - loss: 0.2070 - accuracy: 0.9817 - val_loss: 0.211
9 - val_accuracy: 0.9776 - lr: 5.0000e-04
Epoch 23/30
235/235 [==============================] - 620s 3s/step - loss: 0.1731 - accuracy: 0.9920 - val_loss: 0.234
7 - val_accuracy: 0.9723 - lr: 2.5000e-04
Epoch 24/30
235/235 [==============================] - 624s 3s/step - loss: 0.1670 - accuracy: 0.9923 - val_loss: 0.489
3 - val_accuracy: 0.8640 - lr: 2.5000e-04
Epoch 25/30
235/235 [==============================] - 618s 3s/step - loss: 0.1607 - accuracy: 0.9917 - val_loss: 0.187
1 - val_accuracy: 0.9835 - lr: 2.5000e-04
Epoch 26/30
235/235 [==============================] - 617s 3s/step - loss: 0.1566 - accuracy: 0.9927 - val_loss: 0.340
6 - val_accuracy: 0.9371 - lr: 2.5000e-04
Epoch 27/30
235/235 [==============================] - 617s 3s/step - loss: 0.1506 - accuracy: 0.9936 - val_loss: 0.246
0 - val_accuracy: 0.9605 - lr: 2.5000e-04
Epoch 28/30
```

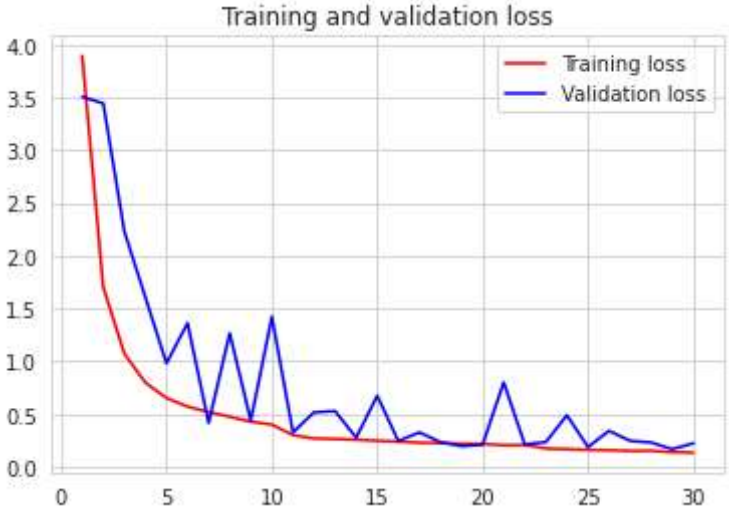
```
235/235 [==============================] - ETA: 0s - loss: 0.1512 - accuracy: 0.9917
Epoch 28: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
235/235 [==============================] - 617s 3s/step - loss: 0.1512 - accuracy: 0.9917 - val_loss: 0.230
2 - val_accuracy: 0.9637 - lr: 2.5000e-04
Epoch 29/30
235/235 [==============================] - 620s 3s/step - loss: 0.1387 - accuracy: 0.9941 - val_loss: 0.166
9 - val_accuracy: 0.9851 - lr: 1.2500e-04
Epoch 30/30
235/235 [==============================] - 622s 3s/step - loss: 0.1350 - accuracy: 0.9960 - val_loss: 0.223
7 - val_accuracy: 0.9680 - lr: 1.2500e-04
```

In [ ]:

```
acc = history1.history['accuracy']
val_acc = history1.history['val_accuracy']
loss = history1.history['loss']
val_loss = history1.history['val_loss']
epochs = range(1, len(acc) + 1)

sns.set_style("whitegrid")
plt.title('Training and Validation Accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```

```
<IPython.core.display.Javascript object>
```





In [ ]:

```
1 Y_pred = net1.predict(validation_generator)
2 y_pred = np.argmax(Y_pred, axis=1)
3 report1 = classification_report(validation_generator.classes, y_pred, target_names=rice_classes, output_
4 df1 = pd.DataFrame(report1).transpose()
5 df1
```

<IPython.core.display.Javascript object>

Out[31]:

	precision	recall	f1-score	support
Arborio	1.000000	0.853333	0.920863	375.000
Basmati	0.989446	1.000000	0.994695	375.000
Ipsala	0.968992	1.000000	0.984252	375.000
Jasmine	0.894231	0.992000	0.940582	375.000
Karacadag	1.000000	0.994667	0.997326	375.000
accuracy	0.968000	0.968000	0.968000	0.968
macro avg	0.970534	0.968000	0.967544	1875.000
weighted avg	0.970534	0.968000	0.967544	1875.000

It can be seen from the classification report that the model has very good classification goodness-of-fit for all classes of the target variable.
In particular, the model **doesn't overfit** in the last epochs, aproximately from the 23th epoch. The overall **accuracy** on the validation set is high and equal to 0.968.
Finally, we can note that the model is stable in the last 10/15 epochs.
It can be said that rice grain images are often correctly classified.

In the cell below we save the model.

In [ ]:
```
# Saving

net1.save('/content/gdrive/MyDrive/FDL2022Project/modelli ve/BestModelFromScratch.h5')
```

```
<IPython.core.display.Javascript object>
```

###5.2) **Best Model with Transfer Learning**

To achieve better performance than those obtained on the networks previously developed we use **Transfer learning** (fine tuning) approach. The best pre-trained netework selected from **Notebook 2** is the ResNet50.

In [ ]:
```
pretrained_model = resnet50.ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(size, size, 3),
    pooling='avg',
)
```

```
<IPython.core.display.Javascript object>
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5)
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

We want to freeze the weights that we "inherited" from the pretraining on imagenet:

In [ ]:
```
layer_dict = dict([(layer.name, layer) for layer in pretrained_model.layers])

for layer in pretrained_model.layers:
    layer.trainable = False
```

```
<IPython.core.display.Javascript object>
```

Now we can definie the network architecture as follow:

- Dense layer 2048 neurons, activation function "relu".
- Dense layer 512 neurons, activation function "relu".
- Dropout layer 0.15.
- Dense layer 128 neurons, activation function "relu".
- Dense layer 64 neurons, activation function "relu".
- Dropout layer 0.15.
- Dense layer 32 neurons, activation function "relu".
- Dense layer 16 neurons, activation function "relu".
- Output layer 5 output neurons, activation function "softmax".

In [ ]:
```
x2 = pretrained_model.output
x2 = Dense(2048,'relu')(x2)
x2 = Dense(512,'relu')(x2)
x2 = Dropout(.15)(x2)
x2 = Dense(128,'relu')(x2)
x2 = Dense(64,'relu')(x2)
x2 = Dropout(.15)(x2)
x2 = Dense(32,'relu')(x2)
x2 = Dense(16,'relu')(x2)
pred2 = Dense(5,'softmax')(x2)

model2 =  Model(inputs=pretrained_model.input, outputs=pred2)
```

```
<IPython.core.display.Javascript object>
```

A summary of the neural network architecture described above is shown in the cells below. In particular, we note that the total number of parameters to be estimated is 28 909 765 of which 5 322 053 are trainable while 23 587 712 are not. In particular, the parameter not trainable are the ones from the ResNet50.

In [ ]:
```
1 model2.summary()
```

```
<IPython.core.display.Javascript object>

Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_1[0][0]']
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 112, 112, 64)	0	['conv1_bn[0][0]']

In [ ]:
```
1 model2.compile(
2   optimizer=tf.keras.optimizers.Adam(lr=0.0001),
3   loss='categorical_crossentropy',
4   metrics=['accuracy'])
```

```
<IPython.core.display.Javascript object>

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is de
precated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:

```
EarlyStopping=tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=0,
    patience=3,
    verbose=0,
    mode='auto',
    baseline=None,
    restore_best_weights=True
)
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

```
history = model2.fit_generator(
    generator=train_generator_tf,
    epochs=25,
    validation_data=validation_generator_tf,
    callbacks=[EarlyStopping])
```

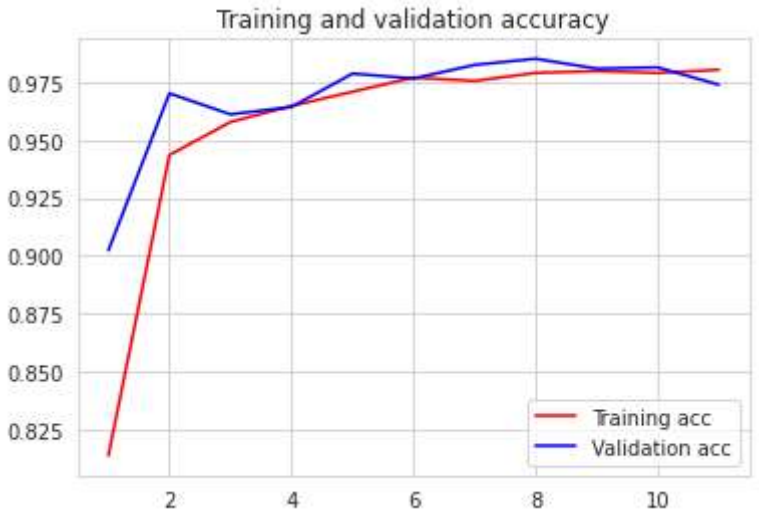
```
<IPython.core.display.Javascript object>

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Model.fit_generator` is depre
cated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  """
```

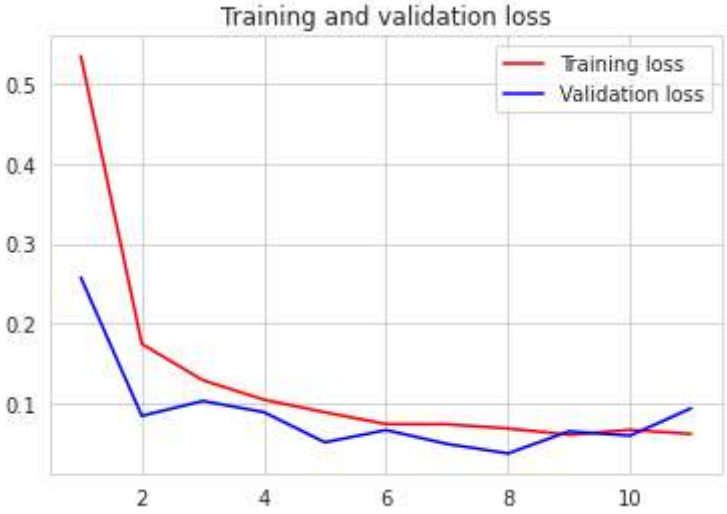
```
Epoch 1/25
118/118 [==============================] - 1994s 17s/step - loss: 0.5351 - accuracy: 0.8139 - val_loss: 0.2
574 - val_accuracy: 0.9024
Epoch 2/25
118/118 [==============================] - 108s 913ms/step - loss: 0.1743 - accuracy: 0.9435 - val_loss: 0.
0841 - val_accuracy: 0.9701
Epoch 3/25
118/118 [==============================] - 107s 907ms/step - loss: 0.1288 - accuracy: 0.9578 - val_loss: 0.
1028 - val_accuracy: 0.9611
Epoch 4/25
118/118 [==============================] - 108s 913ms/step - loss: 0.1044 - accuracy: 0.9647 - val_loss: 0.
0887 - val_accuracy: 0.9643
Epoch 5/25
118/118 [==============================] - 107s 907ms/step - loss: 0.0886 - accuracy: 0.9708 - val_loss: 0.
0510 - val_accuracy: 0.9787
Epoch 6/25
118/118 [==============================] - 107s 908ms/step - loss: 0.0739 - accuracy: 0.9769 - val_loss: 0.
0663 - val_accuracy: 0.9765
Epoch 7/25
118/118 [==============================] - 109s 920ms/step - loss: 0.0736 - accuracy: 0.9755 - val_loss: 0.
0490 - val_accuracy: 0.9824
Epoch 8/25
118/118 [==============================] - 108s 917ms/step - loss: 0.0683 - accuracy: 0.9789 - val_loss: 0.
0371 - val_accuracy: 0.9851
Epoch 9/25
118/118 [==============================] - 107s 906ms/step - loss: 0.0605 - accuracy: 0.9797 - val_loss: 0.
0648 - val_accuracy: 0.9808
Epoch 10/25
118/118 [==============================] - 108s 911ms/step - loss: 0.0666 - accuracy: 0.9789 - val_loss: 0.
0593 - val_accuracy: 0.9813
Epoch 11/25
118/118 [==============================] - 107s 907ms/step - loss: 0.0615 - accuracy: 0.9803 - val_loss: 0.
0937 - val_accuracy: 0.9739
```

In [ ]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

sns.set_style("whitegrid")
plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```

```
<IPython.core.display.Javascript object>
```





In [ ]:
```
1 model2.save('/content/gdrive/MyDrive/FDL2022Project/modelli ve/BestTransfer.h5')
```

```
<IPython.core.display.Javascript object>
```

In [ ]:
```
Y_pred = model2.predict(validation_generator_tf)
y_pred = np.argmax(Y_pred, axis=1)
report1 = classification_report(validation_generator_tf.classes, y_pred, target_names=rice_classes, outp
df1 = pd.DataFrame(report1).transpose()
df1
```

```
<IPython.core.display.Javascript object>
```

Out[28]:

	precision	recall	f1-score	support
Arborio	0.971204	0.989333	0.980185	375.000000
Basmati	0.981627	0.997333	0.989418	375.000000
Ipsala	0.992021	0.994667	0.993342	375.000000
Jasmine	0.991690	0.954667	0.972826	375.000000
Karacadag	0.989333	0.989333	0.989333	375.000000
accuracy	0.985067	0.985067	0.985067	0.985067
macro avg	0.985175	0.985067	0.985021	1875.000000
weighted avg	0.985175	0.985067	0.985021	1875.000000

The result is very satisfactory, the **validation accuracy** achieved is 98%, and the network never overfit: **validation loss** is always under the Training curve, the same is for the accuracy.

6. Test

To finally choose the best model, we created a new dataset, called **test** on which we fitted the two models.

The test dataset contains 100 images per class, consequently it contains 500 images.

In [ ]:
```
# Draw sample from the original dataset
from random import randint

# Index for Random Sampling
index = []
for _ in range(100):
value = randint(0, 15000)
index.append(value)

#Define source dir and destination dir
src_dir = '/content/gdrive/MyDrive/FDL2022Project/Dati/Original/Rice_Image_Dataset/'
dst_dir = '/content/gdrive/MyDrive/FDL2022Project/Dati/Sample/Test/'

import random
random.seed(123)
random_sampling(index, src_dir=src_dir, dst_dir=dst_dir)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

In [ ]:
```
test_path = '/content/gdrive/MyDrive/FDL2022Project/Dati/Sample/Test/'
```

```
<IPython.core.display.Javascript object>
```

6.1) Preprocessing and Splitting into Training and Validation for Best Model from scratch

In [ ]:
```
from keras.preprocessing.image import ImageDataGenerator
```

```
<IPython.core.display.Javascript object>
```

In [ ]:
```
# Validation dataset: standardization

test_datagen1 = ImageDataGenerator(rescale=1/255,
                                   validation_split=0.2)
```

```
<IPython.core.display.Javascript object>
```

In [ ]:
```
# Training data loader

test_generator1 = test_datagen1.flow_from_directory(test_path,
                                                    target_size=(224,224),
                                                    color_mode='rgb',
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    shuffle=False,
                                                    seed=1)

```

```
<IPython.core.display.Javascript object>

Found 500 images belonging to 5 classes.
```

In [ ]:
```
# net_test_BfS = keras.models.load_model('/content/gdrive/MyDrive/FDL2022Project/modelli ve/BestModelFro
```

In [ ]:
```
rice_classes = os.listdir(test_path)
print(rice_classes)
```

```
<IPython.core.display.Javascript object>

['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

In [ ]:
```
Y_pred_testBfS = net1.predict(test_generator1)
y_pred_testBfS = np.argmax(Y_pred_testBfS, axis=1)
report_testBfS  = classification_report(test_generator1.classes, y_pred_testBfS, target_names=rice_class
df1_testBfS = pd.DataFrame(report_testBfS).transpose()
df1_testBfS
```

<IPython.core.display.Javascript object>

Out[46]:

	precision	recall	f1-score	support
Arborio	1.000000	0.860	0.924731	100.000
Basmati	0.980392	1.000	0.990099	100.000
Ipsala	0.943396	1.000	0.970874	100.000
Jasmine	0.915888	0.980	0.946860	100.000
Karacadag	1.000000	0.990	0.994975	100.000
accuracy	0.966000	0.966	0.966000	0.966
macro avg	0.967935	0.966	0.965508	500.000
weighted avg	0.967935	0.966	0.965508	500.000

6.2) Preprocessing and Splitting into Training and Validation for Best Model with Transfer Learning

In [ ]:
```
test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input_resnet50)
```

<IPython.core.display.Javascript object>

In [ ]:
```
size=224
channels=3
batch_size = 64
num_classes = 5
```

<IPython.core.display.Javascript object>

In [ ]:
```
test_generator_tf = test_datagen.flow_from_directory(test_path,
                                                     target_size=(size,size),
                                                     color_mode="rgb",
                                                     batch_size=batch_size,
                                                     class_mode='categorical',
                                                     shuffle=False,
                                                     interpolation='nearest')
```

<IPython.core.display.Javascript object>

Found 500 images belonging to 5 classes.

In [ ]:
```
from tensorflow import keras
model2 = keras.models.load_model('/content/gdrive/MyDrive/FDL2022Project/modelli ve/BestTransfer.h5')
```

<IPython.core.display.Javascript object>

In [ ]:
```
Y_pred = model2.predict(test_generator_tf)
y_pred = np.argmax(Y_pred, axis=1)
report2 = classification_report(test_generator_tf.classes, y_pred, target_names=rice_classes, output_dic
df2 = pd.DataFrame(report2).transpose()
df2
```

<IPython.core.display.Javascript object>

Out[17]:

	precision	recall	f1-score	support
Arborio	0.970874	1.000	0.985222	100.000
Basmati	0.961538	1.000	0.980392	100.000
Ipsala	1.000000	1.000	1.000000	100.000
Jasmine	1.000000	0.950	0.974359	100.000
Karacadag	1.000000	0.980	0.989899	100.000
accuracy	0.986000	0.986	0.986000	0.986
macro avg	0.986482	0.986	0.985974	500.000
weighted avg	0.986482	0.986	0.985974	500.000