Neural network classifier for mushrooms

In [1]:

```
import os
import shutil
import pandas as pd

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.python.framework.config import list_physical_devices, set_memory_growth
```

In [2]:

```
# To fix "Image File is truncated" error during training
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

In [3]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
```

In [4]:

```
# Settings for displaying charts
plt.rcParams['figure.figsize'] = 12, 8
plt.rcParams.update({'font.size': 12})
```

In [5]:

```
physical_devices = list_physical_devices('GPU')
print(f'Number of GPUs available: {len(physical_devices)}')

if len(physical_devices) > 0:
    set_memory_growth(physical_devices[0], True)
```

```
Number of GPUs available: 1
```

In [6]:

```
# TensorFlow settings
AUTOTUNE = tf.data.experimental.AUTOTUNE
IMG_SIZE = 299
BATCH_SIZE = 64
```

Loading and processing data

In [7]:

```
# Path to the folder with 9 classes of images:
data_path = '/kaggle/input/mushrooms-classification-common-genuss-images/Mushrooms'
```

In [8]:

```
# Temporary folders for training, validation and test images:
os.mkdir('/kaggle/temp')
os.chdir('/kaggle/temp')
os.mkdir('train')
os.mkdir('valid')
os.mkdir('test')
os.chdir('/kaggle/working')
```

In [9]:

```
# Split images (75%/15%/10%) and save to temporary folders:
for subfolder in os.listdir(data_path):

    # Making a list of all files in current subfolder:
    original_path = f'{data_path}/{subfolder}'
    original_data = os.listdir(original_path)

    # Number of samples in each group:
    n_samples = len(original_data)
    train_samples = int(n_samples * 0.75)
    valid_samples = int(n_samples * 0.9)

    train_path = f'/kaggle/temp/train/{subfolder}'
    valid_path = f'/kaggle/temp/valid/{subfolder}'
    test_path = f'/kaggle/temp/test/{subfolder}'

    # New class subfolder for training:
    os.chdir('/kaggle/temp/train')
    os.mkdir(subfolder)

    # Training images:
    for image in range(train_samples):
        original_file = f'{original_path}/{original_data[image]}'
        new_file = f'{train_path}/{original_data[image]}'
        shutil.copyfile(original_file, new_file)

    # New class subfolder for validation:
    os.chdir('/kaggle/temp/valid')
    os.mkdir(subfolder)

    # Validation images:
    for image in range(train_samples, valid_samples):
        original_file = f'{original_path}/{original_data[image]}'
        new_file = f'{valid_path}/{original_data[image]}'
        shutil.copyfile(original_file, new_file)

    # New class subfolder for testing:
    os.chdir('/kaggle/temp/test')
    os.mkdir(subfolder)

    # Test images:
    for image in range(valid_samples, n_samples):
        original_file = f'{original_path}/{original_data[image]}'
        new_file = f'{test_path}/{original_data[image]}'
        shutil.copyfile(original_file, new_file)
```

In [10]:

```
# Displaying examples from each class
nrows = 3
ncols = 3

pos = 0

for subfolder in os.listdir(data_path):

    image_file = os.listdir(os.path.join(data_path, subfolder))[0]

    fig = plt.gcf()
    fig.set_size_inches(ncols * 4, nrows * 4)

    pos += 1
    sp = plt.subplot(nrows, ncols, pos)

    cur_image = mpimg.imread(os.path.join(data_path, subfolder, image_file))
    plt.imshow(cur_image)
    plt.title(subfolder)
    plt.axis('Off')
```

Creating a model

In [12]:

```
# Pretrained EfficientNetB7 image classification model without final layers
feature_model = tf.keras.applications.EfficientNetB7(weights='imagenet',
                                                     include_top=False,
                                                     input_shape=(IMG_SIZE, IMG_SIZE, 3),
                                                     pooling='avg')

feature_model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb7_n
otop.h5
258080768/258076736 [==============================] - 1s 0us/step
Model: "efficientnetb7"
__________________________________________________________________________________________
__________
Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
==========
input_1 (InputLayer)           [(None, 299, 299, 3) 0
__________________________________________________________________________________________
__________
rescaling (Rescaling)          (None, 299, 299, 3)  0           input_1[0][0]
__________________________________________________________________________________________
__________
normalization (Normalization)  (None, 299, 299, 3)  7           rescaling[0][0]
__________________________________________________________________________________________
__________
stem_conv_pad (ZeroPadding2D)  (None, 301, 301, 3)  0           normalization[0][0]
__________________________________________________________________________________________
__________
stem_conv (Conv2D)             (None, 150, 150, 64) 1728        stem_conv_pad[0][0]
__________________________________________________________________________________________
__________
stem_bn (BatchNormalization)   (None, 150, 150, 64) 256         stem_conv[0][0]
__________________________________________________________________________________________
__________
stem_activation (Activation)   (None, 150, 150, 64) 0           stem_bn[0][0]
__________________________________________________________________________________________
__________
block1a_dwconv (DepthwiseConv2D (None, 150, 150, 64) 576        stem_activation[0][0]
__________________________________________________________________________________________
__________
block1a_bn (BatchNormalization) (None, 150, 150, 64) 256        block1a_dwconv[0][0]
__________________________________________________________________________________________
__________
block1a_activation (Activation) (None, 150, 150, 64) 0          block1a_bn[0][0]
__________________________________________________________________________________________
__________
block1a_se_squeeze (GlobalAvera (None, 64)           0           block1a_activation[0]
[0]
__________________________________________________________________________________________
__________
block1a_se_reshape (Reshape)   (None, 1, 1, 64)     0           block1a_se_squeeze[0]
[0]
__________________________________________________________________________________________
```

In [13]:

```
# Construct a new model with the final dense layer for 9 classes
new_model = tf.keras.models.Sequential(
    [
        feature_model,
        tf.keras.layers.Dense(9, activation='softmax')
    ]
)
```

In [14]:

```
# Make all the layers from the original ResNet model untrainable
new_model.layers[0].trainable = False
```

In [15]:

```
# Metrics and optimizer
new_model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

In [16]:

```
# Check the architecture of the new model
new_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb7 (Functional)  (None, 2560)              64097687
_________________________________________________________________
dense (Dense)                (None, 9)                 23049
=================================================================
Total params: 64,120,736
Trainable params: 23,049
Non-trainable params: 64,097,687
_________________________________________________________________
```

In [17]:

```
# Callbacks to be exercised during training
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                              patience=10,
                                              restore_best_weights=True)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                                 factor=0.1,
                                                 mode='max',
                                                 cooldown=2,
                                                 patience=2,
                                                 min_lr=0)
```

In [18]:

```python
# Train new model:
history = new_model.fit(train_generator,
                        validation_data=valid_generator,
                        epochs=100,
                        # steps_per_epoch = num_train_samples (5033) / batch size (64)
                        steps_per_epoch=79,
                        # validation_steps = num_validation_samples (1005) / batch size (64)
                        validation_steps=16,
                        verbose=2,
                        callbacks=[reduce_lr, early_stop],
                        use_multiprocessing=True,
                        workers=2)
```

```
Epoch 1/100
79/79 - 102s - loss: 1.4300 - accuracy: 0.5186 - val_loss: 1.1327 - val_accuracy: 0.6398
Epoch 2/100
79/79 - 97s - loss: 1.0150 - accuracy: 0.6743 - val_loss: 0.9715 - val_accuracy: 0.6856
Epoch 3/100
79/79 - 100s - loss: 0.8807 - accuracy: 0.7222 - val_loss: 0.8767 - val_accuracy: 0.7274
Epoch 4/100
79/79 - 99s - loss: 0.7953 - accuracy: 0.7499 - val_loss: 0.8204 - val_accuracy: 0.7284
Epoch 5/100
79/79 - 98s - loss: 0.7350 - accuracy: 0.7681 - val_loss: 0.7830 - val_accuracy: 0.7274
Epoch 6/100
79/79 - 101s - loss: 0.6927 - accuracy: 0.7799 - val_loss: 0.7598 - val_accuracy: 0.7443
Epoch 7/100
79/79 - 99s - loss: 0.6648 - accuracy: 0.7880 - val_loss: 0.7390 - val_accuracy: 0.7552
Epoch 8/100
79/79 - 99s - loss: 0.6272 - accuracy: 0.8043 - val_loss: 0.7128 - val_accuracy: 0.7592
Epoch 9/100
79/79 - 101s - loss: 0.5991 - accuracy: 0.8136 - val_loss: 0.6951 - val_accuracy: 0.7652
Epoch 10/100
79/79 - 100s - loss: 0.5713 - accuracy: 0.8216 - val_loss: 0.6866 - val_accuracy: 0.7741
Epoch 11/100
79/79 - 99s - loss: 0.5576 - accuracy: 0.8273 - val_loss: 0.6811 - val_accuracy: 0.7682
Epoch 12/100
79/79 - 100s - loss: 0.5343 - accuracy: 0.8339 - val_loss: 0.6663 - val_accuracy: 0.7811
Epoch 13/100
79/79 - 98s - loss: 0.5133 - accuracy: 0.8454 - val_loss: 0.6531 - val_accuracy: 0.7662
Epoch 14/100
79/79 - 99s - loss: 0.5044 - accuracy: 0.8484 - val_loss: 0.6423 - val_accuracy: 0.7791
Epoch 15/100
79/79 - 99s - loss: 0.4859 - accuracy: 0.8514 - val_loss: 0.6383 - val_accuracy: 0.7801
Epoch 16/100
79/79 - 99s - loss: 0.4847 - accuracy: 0.8544 - val_loss: 0.6382 - val_accuracy: 0.7791
Epoch 17/100
79/79 - 99s - loss: 0.4826 - accuracy: 0.8530 - val_loss: 0.6365 - val_accuracy: 0.7781
Epoch 18/100
79/79 - 99s - loss: 0.4778 - accuracy: 0.8581 - val_loss: 0.6363 - val_accuracy: 0.7781
Epoch 19/100
79/79 - 99s - loss: 0.4793 - accuracy: 0.8556 - val_loss: 0.6362 - val_accuracy: 0.7781
Epoch 20/100
79/79 - 99s - loss: 0.4856 - accuracy: 0.8494 - val_loss: 0.6361 - val_accuracy: 0.7791
Epoch 21/100
79/79 - 99s - loss: 0.4780 - accuracy: 0.8526 - val_loss: 0.6361 - val_accuracy: 0.7791
Epoch 22/100
79/79 - 100s - loss: 0.4832 - accuracy: 0.8554 - val_loss: 0.6361 - val_accuracy: 0.7791
```

Displaying the results

Displaying the results

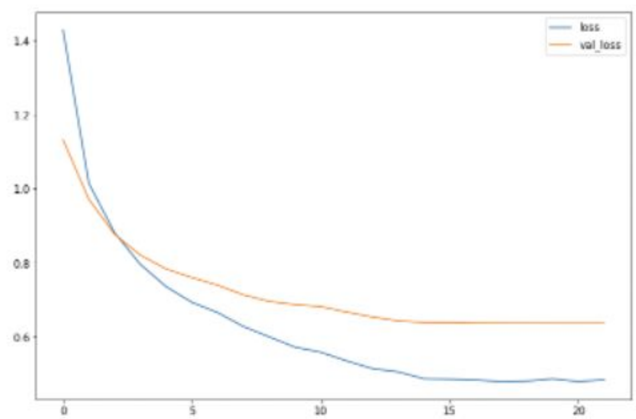
In [19]:

```
loss, accuracy = new_model.evaluate(test_generator,
                                    steps=11,
                                    verbose=2,
                                    use_multiprocessing=True,
                                    workers=2)
print(f'Model performance on test images:\nAccuracy = {accuracy}\nLoss = {loss}')
```

```
11/11 - 12s - loss: 0.6501 - accuracy: 0.7973
Model performance on test images:
Accuracy = 0.7973372936248779
Loss = 0.6501479148864746
```

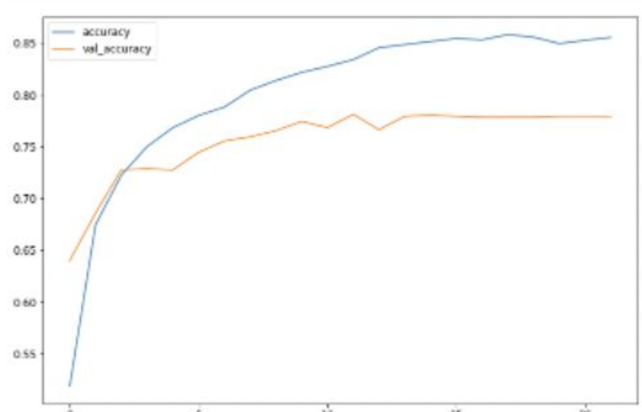
In [20]:

```
# Loss during training:
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot();
```



In [21]:

```
# Accuracy during training:
history_frame.loc[:, ['accuracy', 'val_accuracy']].plot();
```



In [22]:

```
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

nb_samples = 676 # number of test images
Y_pred = new_model.predict_generator(test_generator, nb_samples // BATCH_SIZE+1)
y_pred = np.argmax(Y_pred, axis=1)

print('Confusion Matrix')
print(confusion_matrix(test_generator.classes, y_pred))
# x is true class, y is predicted class-- middle diagonal represents the accurate predictions
```

```
Confusion Matrix
[[ 20   5   1   2   0   0   3   4   1]
 [  5  63   1   2   1   0   1   2   0]
 [  0   2 102   0   0   0   0   2   2]
 [  0   1   2  63   2   0  10   3   3]
 [  1   0   0   3  29   1   0   3   0]
 [  0   0   0   0   2  26   3   1   0]
 [  2   4   2   3   4   0 127  14   1]
 [  0   0   1   3   0   0  21  90   0]
 [  0   1   3   5   0   0   4   0  19]]
```

In [23]:

```
print('Classification Report')
target_names = list(train_generator.class_indices.keys())
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

```
Classification Report
              precision    recall  f1-score   support

    Agaricus       0.71      0.56      0.63        36
     Amanita       0.83      0.84      0.83        75
     Boletus       0.91      0.94      0.93       108
 Cortinarius       0.78      0.75      0.76        84
    Entoloma       0.76      0.78      0.77        37
   Hygrocybe       0.96      0.81      0.88        32
   Lactarius       0.75      0.81      0.78       157
     Russula       0.76      0.78      0.77       115
     Suillus       0.73      0.59      0.66        32

    accuracy                           0.80       676
   macro avg       0.80      0.76      0.78       676
weighted avg       0.80      0.80      0.80       676
```

In [ ]:

In [ ]: