Project2

November 19, 2023

[1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, BatchNormalization
from tensorflow.keras.optimizers.legacy import Adam

from datetime import datetime

# Set seed
np.random.seed(42)
tf.random.set_seed(42)

# plt style
plt.style.use('ggplot')
```

[2]:
```
# Data directories
train_dir = 'Dataset2/FNA'
test_dir = 'Dataset2/test'
```

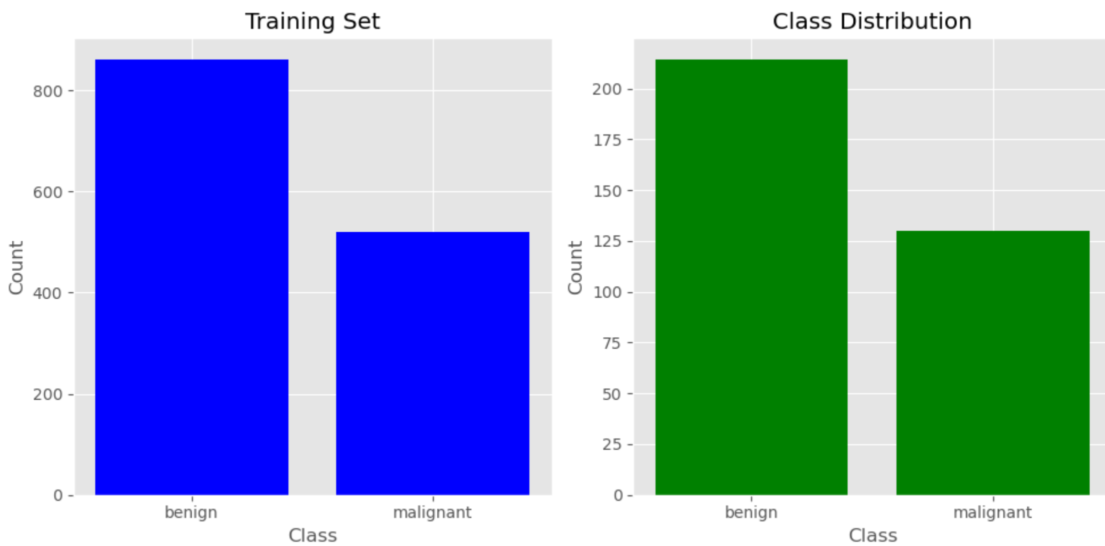
[3]:
```
# Preprocessing and data augmentation
train_datagen = ImageDataGenerator(
    rescale=1.0/255,     # Rescale input pixel values to [0,1]
    validation_split=0.2   # Split training data into 80% training and 20%␣
↪validation
)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(50, 50),
    batch_size=32,
    class_mode='binary',
    subset='training'
```

```
)
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(50, 50),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 1380 images belonging to 2 classes.
Found 344 images belonging to 2 classes.
```

[4]:

```
# Class names
class_names = list(train_generator.class_indices.keys())
# Plot Class distribution in training and validation sets with class names
train_value, train_counts = np.unique(
    train_generator.classes, return_counts=True)
validation_value, validation_counts = np.unique(
    validation_generator.classes, return_counts=True)

fg, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].bar(class_names, train_counts, color='blue')
ax[0].set_title('Training Set')
ax[0].set_xlabel('Class')
ax[0].set_ylabel('Count')
ax[1].bar(class_names, validation_counts, color='green')
ax[1].set_title('Validation Set')
ax[1].set_xlabel('Class')
ax[1].set_ylabel('Count')
plt.title('Class Distribution')
plt.tight_layout()
plt.show()
```



[5]:

```
# CNN model
model = Sequential()

# Convolutional layer 1
model.add(Conv2D(32, (3, 3), activation='relu',
          input_shape=(50, 50, 3), padding='same'))
# Normalize the activations of the previous layer at each batch
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Convolutional layer 2
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Convolutional layer 3
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Convolutional layer 4
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Flatten the feature maps
model.add(Flatten())
```

```python
# Fully connected layer 1
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))  # Regularization to prevent overfitting

# Output layer
model.add(Dense(1, activation='sigmoid'))  # Sigmoid for binary classification

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

```
2023-11-19 23:20:03.171771: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M2 Pro
2023-11-19 23:20:03.171803: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2023-11-19 23:20:03.171817: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
2023-11-19 23:20:03.171856: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:306]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2023-11-19 23:20:03.171877: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 batch_normalization (Batch  (None, 50, 50, 32)        128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 25, 25, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 25, 25, 64)        18496

 batch_normalization_1 (Bat  (None, 25, 25, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)
```

```
 conv2d_2 (Conv2D)           (None, 12, 12, 128)       73856

 batch_normalization_2 (Bat  (None, 12, 12, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 6, 6, 128)         147584

 batch_normalization_3 (Bat  (None, 6, 6, 128)         512
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 128)         0
 g2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 389953 (1.49 MB)
Trainable params: 389249 (1.48 MB)
Non-trainable params: 704 (2.75 KB)
_________________________________________________________________
```

---

[6]:

```
# Define a ReduceLROnPlateau callback
lr_optimization = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    min_delta=1e-7,
    cooldown=0,
)

# Early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=10,
    min_delta=1e-7,
    restore_best_weights=True
```

```
)


# Training
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=100,
    verbose=1,
    callbacks=[
        lr_optimization,
        early_stopping
    ]
)
```

```
Epoch 1/100
```

```
2023-11-19 23:20:03.762638: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
```

```
44/44 [==============================] - 2s 25ms/step - loss: 0.4737 - accuracy:
0.8703 - val_loss: 0.7446 - val_accuracy: 0.3779 - lr: 1.0000e-04
Epoch 2/100
44/44 [==============================] - 1s 15ms/step - loss: 0.2991 - accuracy:
0.9051 - val_loss: 1.0794 - val_accuracy: 0.3779 - lr: 1.0000e-04
Epoch 3/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1871 - accuracy:
0.9449 - val_loss: 1.2693 - val_accuracy: 0.3779 - lr: 1.0000e-04
Epoch 4/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1356 - accuracy:
0.9565 - val_loss: 1.7495 - val_accuracy: 0.3779 - lr: 2.0000e-05
Epoch 5/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1511 - accuracy:
0.9500 - val_loss: 2.0931 - val_accuracy: 0.3779 - lr: 2.0000e-05
Epoch 6/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1170 - accuracy:
0.9580 - val_loss: 2.3287 - val_accuracy: 0.3779 - lr: 4.0000e-06
Epoch 7/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1164 - accuracy:
0.9594 - val_loss: 2.4348 - val_accuracy: 0.3779 - lr: 4.0000e-06
Epoch 8/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1355 - accuracy:
0.9551 - val_loss: 2.2894 - val_accuracy: 0.3779 - lr: 8.0000e-07
Epoch 9/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1347 - accuracy:
0.9522 - val_loss: 1.9425 - val_accuracy: 0.3750 - lr: 8.0000e-07
Epoch 10/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1217 - accuracy:
```

```
0.9572 - val_loss: 1.5212 - val_accuracy: 0.4302 - lr: 1.6000e-07
Epoch 11/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1193 - accuracy:
0.9594 - val_loss: 1.1397 - val_accuracy: 0.5407 - lr: 1.6000e-07
Epoch 12/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1075 - accuracy:
0.9630 - val_loss: 0.8874 - val_accuracy: 0.6453 - lr: 3.2000e-08
Epoch 13/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1275 - accuracy:
0.9543 - val_loss: 0.7742 - val_accuracy: 0.7297 - lr: 3.2000e-08
Epoch 14/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1198 - accuracy:
0.9572 - val_loss: 0.7310 - val_accuracy: 0.7645 - lr: 6.4000e-09
Epoch 15/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1224 - accuracy:
0.9522 - val_loss: 0.7237 - val_accuracy: 0.7820 - lr: 6.4000e-09
Epoch 16/100
44/44 [==============================] - 1s 15ms/step - loss: 0.1061 - accuracy:
0.9645 - val_loss: 0.7327 - val_accuracy: 0.7791 - lr: 6.4000e-09
Epoch 17/100
44/44 [==============================] - 1s 15ms/step - loss: 0.1098 - accuracy:
0.9572 - val_loss: 0.7459 - val_accuracy: 0.7820 - lr: 6.4000e-09
Epoch 18/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1326 - accuracy:
0.9471 - val_loss: 0.7573 - val_accuracy: 0.7849 - lr: 1.2800e-09
Epoch 19/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1134 - accuracy:
0.9543 - val_loss: 0.7651 - val_accuracy: 0.7936 - lr: 1.2800e-09
Epoch 20/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1240 - accuracy:
0.9493 - val_loss: 0.7749 - val_accuracy: 0.7936 - lr: 2.5600e-10
Epoch 21/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1138 - accuracy:
0.9587 - val_loss: 0.7801 - val_accuracy: 0.7936 - lr: 2.5600e-10
Epoch 22/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1472 - accuracy:
0.9457 - val_loss: 0.7810 - val_accuracy: 0.7965 - lr: 5.1200e-11
Epoch 23/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1325 - accuracy:
0.9558 - val_loss: 0.7858 - val_accuracy: 0.7965 - lr: 5.1200e-11
Epoch 24/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1246 - accuracy:
0.9522 - val_loss: 0.7866 - val_accuracy: 0.7965 - lr: 1.0240e-11
Epoch 25/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1178 - accuracy:
0.9551 - val_loss: 0.7895 - val_accuracy: 0.7965 - lr: 1.0240e-11
Epoch 26/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1160 - accuracy:
```

```
0.9587 - val_loss: 0.7866 - val_accuracy: 0.7965 - lr: 2.0480e-12
Epoch 27/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1215 - accuracy:
0.9522 - val_loss: 0.7863 - val_accuracy: 0.7965 - lr: 2.0480e-12
Epoch 28/100
44/44 [==============================] - 1s 15ms/step - loss: 0.1176 - accuracy:
0.9558 - val_loss: 0.7894 - val_accuracy: 0.7965 - lr: 4.0960e-13
Epoch 29/100
44/44 [==============================] - 1s 17ms/step - loss: 0.1100 - accuracy:
0.9659 - val_loss: 0.7921 - val_accuracy: 0.7965 - lr: 4.0960e-13
Epoch 30/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1262 - accuracy:
0.9543 - val_loss: 0.7920 - val_accuracy: 0.7965 - lr: 8.1920e-14
Epoch 31/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1083 - accuracy:
0.9609 - val_loss: 0.7912 - val_accuracy: 0.7965 - lr: 8.1920e-14
Epoch 32/100
44/44 [==============================] - 1s 16ms/step - loss: 0.1181 - accuracy:
0.9551 - val_loss: 0.7909 - val_accuracy: 0.7965 - lr: 1.6384e-14
```

[ ]:

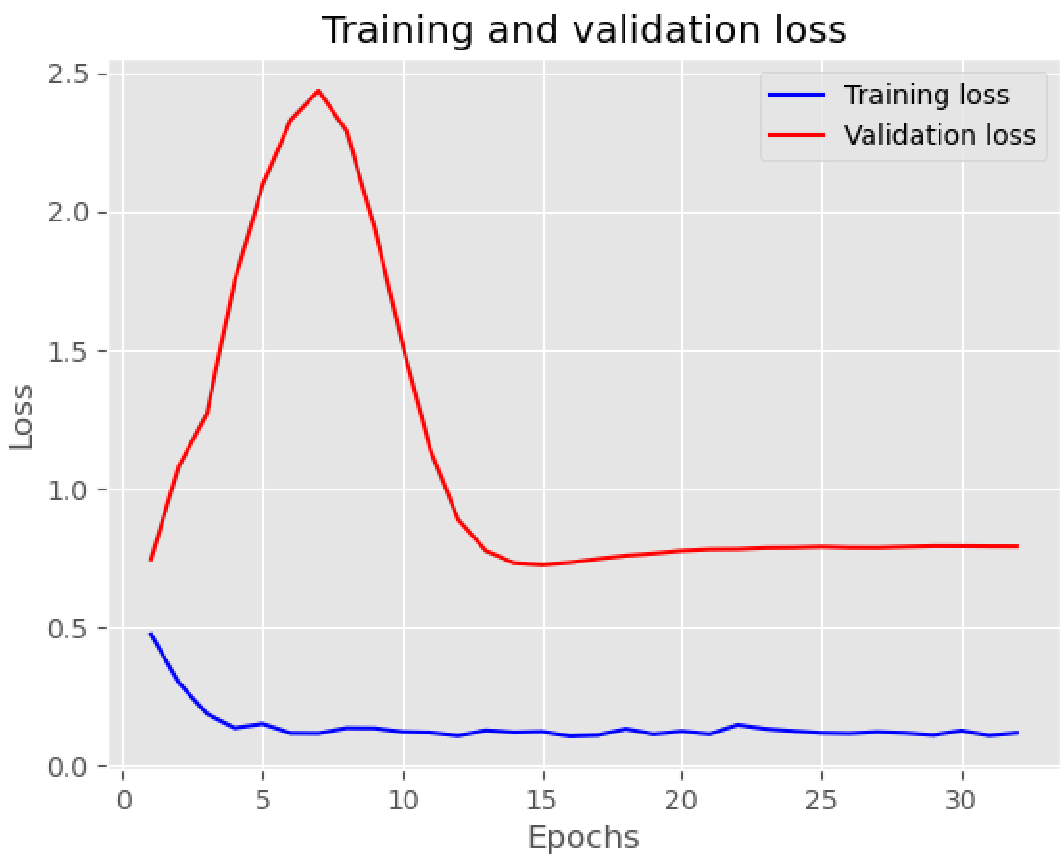
```python
# Save the model
model.save(f'./models/{datetime.now().strftime("%Y_%m_%d-%H_%M_%S")}.keras')
```

[8]:

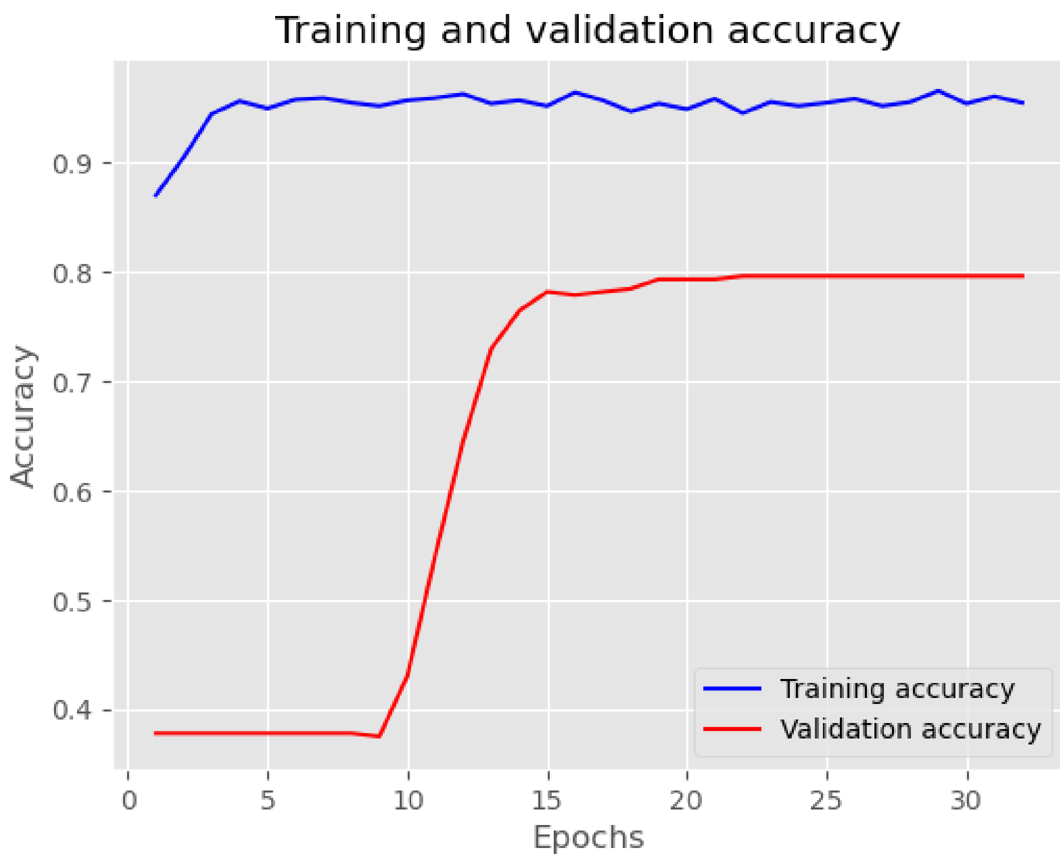
```python
history_dict = history.history
```

[9]:

```python
# Plot the training and validation loss
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values)+1)

plt.plot(epochs, loss_values, 'b', label='Training loss')
plt.plot(epochs, val_loss_values, 'r', label='Validation loss')

plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[10]:
```
# Plot the training and validation accuracy
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']

plt.plot(epochs, acc_values, 'b', label='Training accuracy')
plt.plot(epochs, val_acc_values, 'r', label='Validation accuracy')

plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



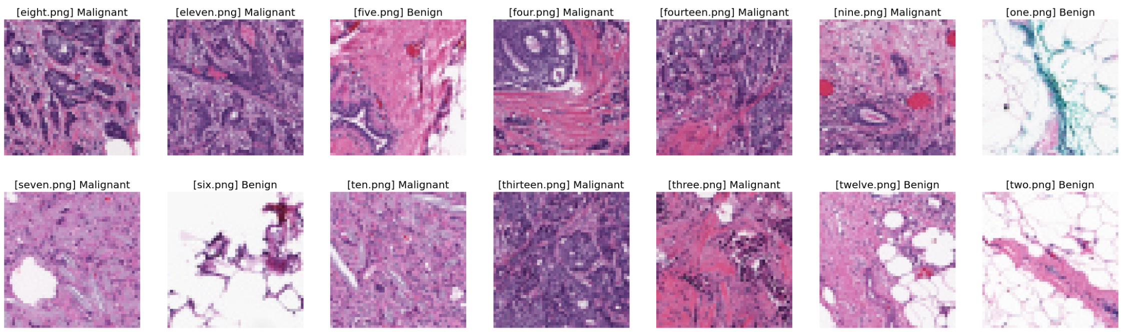
[11]:
```
# Load test data
test_dir = 'Dataset2/test'
os.makedirs(test_dir+'/unknown', exist_ok=True)
for f in os.listdir(test_dir):
    if f.endswith('.png'):
        os.system(
            f'cp {os.path.join(test_dir, f)} {os.path.join(test_dir,␣
↪"unknown")}')

test_datagen = ImageDataGenerator(rescale=1.0/255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(50, 50),
    batch_size=2,
    class_mode=None,
    shuffle=False
)
```

```
Found 14 images belonging to 1 classes.
```

[12]:

```
# Predictions
predictions = model.predict(test_generator)

# Get the filenames from the generator
fnames = test_generator.filenames

# class mapping for the labels in this binary classification problem
index2label = {0: 'benign', 1: 'malignant'}

# Get the corresponding labels
predicted_labels = [index2label[predicted_class]
                    for predicted_class in np.round(predictions.flatten())]

# Plot all the images with their predicted labels
nrows = 2
ncols = len(fnames) // nrows

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

for i, fname in enumerate(fnames):
    sp = plt.subplot(nrows, ncols, i+1)
    sp.axis('Off')

    img = mpimg.imread(os.path.join(test_dir, fname))
    plt.imshow(img)
    plt.title(f'[{fname.split("/")[-1]}] {predicted_labels[i].title()}')

plt.show()
```

```
7/7 [==============================] - 0s 3ms/step
```



[13]:

```
# clean up
ret = os.system(f'rm -rf {test_dir}/unknown')
```