Data Exploration

```
# Importing few libraries
import os
import shutil
import random
from tqdm import tqdm

import numpy as np
import pandas as pd

import PIL
import seaborn as sns
import matplotlib.pyplot as plt
```

```
DATASET = "../input/2750"

LABELS = os.listdir(DATASET)
print(LABELS)
```

```
['SeaLake', 'Highway', 'River', 'Pasture', 'Industrial', 'Residential', 'PermanentCrop', 'AnnualCrop', 'Forest', 'HerbaceousVegetation']
```

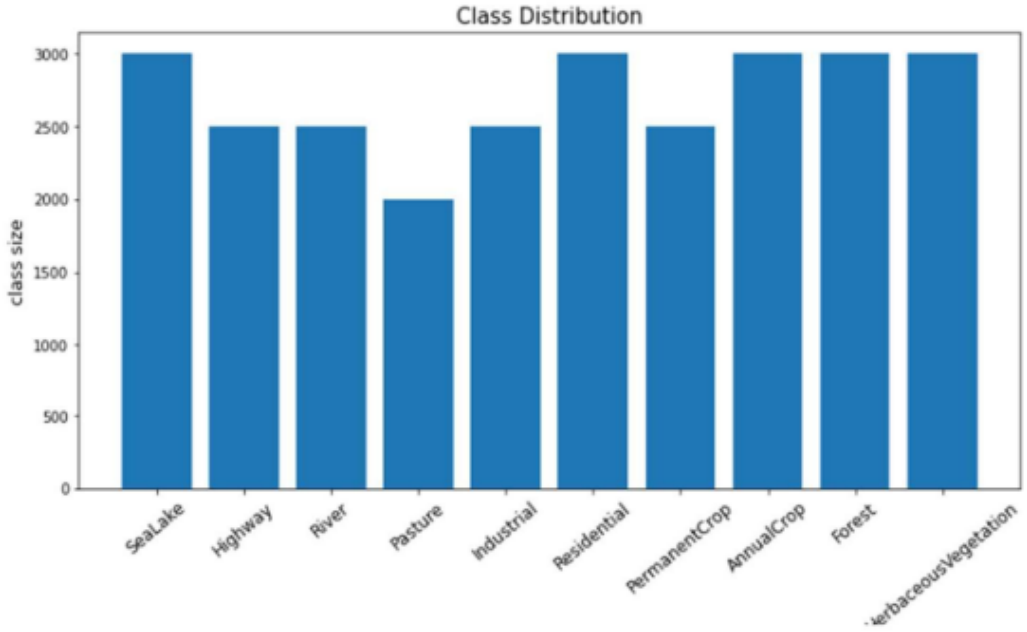
[3]:

```
# plot class distributions of whole dataset
counts = {}

for l in LABELS:
    counts[l] = len(os.listdir(os.path.join(DATASET, l)))


plt.figure(figsize=(12, 6))

plt.bar(range(len(counts)), list(counts.values()), align='center')
plt.xticks(range(len(counts)), list(counts.keys()), fontsize=12, rotation=40)
plt.xlabel('class label', fontsize=13)
plt.ylabel('class size', fontsize=13)
plt.title('Class Distribution', fontsize=15);
```



The dataset is split into 10 classes of land cover. Each class varies in size, so I'll have to stratify later on when splitting the data into training, testing and validation sets.

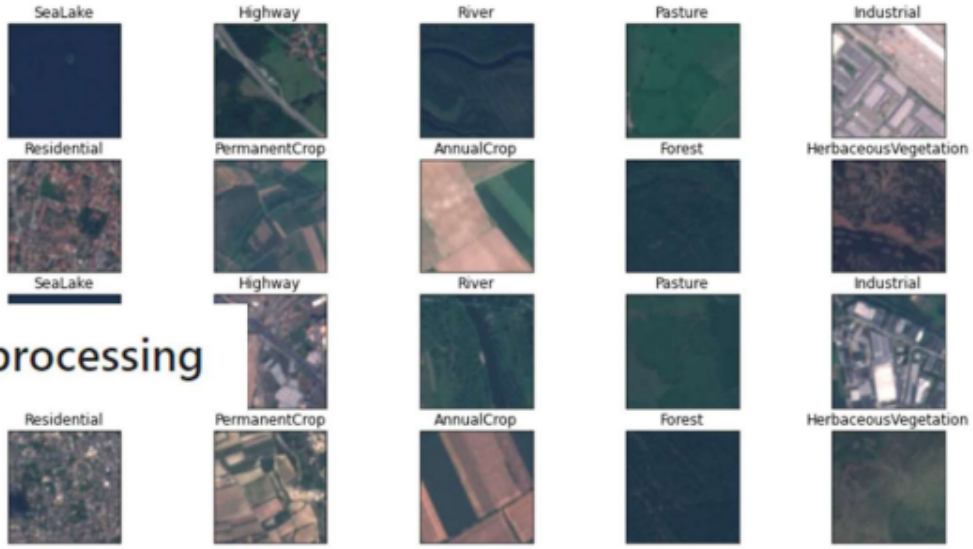
[4]:

```
img_paths = [os.path.join(DATASET, l, l+'_1000.jpg') for l in LABELS]

img_paths = img_paths + [os.path.join(DATASET, l, l+'_2000.jpg') for l in LABELS]

def plot_sat_imgs(paths):
    plt.figure(figsize=(15, 8))
    for i in range(20):
        plt.subplot(4, 5, i+1, xticks=[], yticks=[])
        img = PIL.Image.open(paths[i], 'r')
        plt.imshow(np.asarray(img))
        plt.title(paths[i].split('/')[-2])

plot_sat_imgs(img_paths)
```



Preprocessing

[5]:

```
import re
from sklearn.model_selection import StratifiedShuffleSplit
from keras.preprocessing.image import ImageDataGenerator

TRAIN_DIR = '../working/training'
TEST_DIR = '../working/testing'
BATCH_SIZE = 64
NUM_CLASSES=len(LABELS)
INPUT_SHAPE = (64, 64, 3)
CLASS_MODE = 'categorical'

# create training and testing directories
for path in (TRAIN_DIR, TEST_DIR):
    if not os.path.exists(path):
        os.mkdir(path)

# create class label subdirectories in train and test
for l in LABELS:

    if not os.path.exists(os.path.join(TRAIN_DIR, l)):
        os.mkdir(os.path.join(TRAIN_DIR, l))

    if not os.path.exists(os.path.join(TEST_DIR, l)):
        os.mkdir(os.path.join(TEST_DIR, l))
```

```
Using TensorFlow backend.
```

```
# map each image path to their class label in 'data'
data = {}

for l in LABELS:
    for img in os.listdir(DATASET+'/'+l):
        data.update({os.path.join(DATASET, l, img): l})

X = pd.Series(list(data.keys()))
y = pd.get_dummies(pd.Series(data.values()))

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=69)

# split the list of image paths
for train_idx, test_idx in split.split(X, y):

    train_paths = X[train_idx]
    test_paths = X[test_idx]

    # define a new path for each image depending on training or testing
    new_train_paths = [re.sub('\.\.\/input\/2750', '../working/training', i) for i in train_paths]
    new_test_paths = [re.sub('\.\.\/input\/2750', '../working/testing', i) for i in test_paths]

    train_path_map = list((zip(train_paths, new_train_paths)))
    test_path_map = list((zip(test_paths, new_test_paths)))

    # move the files
    print("moving training files..")
    for i in tqdm(train_path_map):
        if not os.path.exists(i[1]):
            if not os.path.exists(re.sub('training', 'testing', i[1])):
                shutil.copy(i[0], i[1])

    print("moving testing files..")
    for i in tqdm(test_path_map):
        if not os.path.exists(i[1]):
            if not os.path.exists(re.sub('training', 'testing', i[1])):
                shutil.copy(i[0], i[1])
```

```
  0%|          | 14/21600 [00:00<02:43, 131.74it/s]
moving training files..
100%|██████████| 21600/21600 [02:39<00:00, 135.19it/s]
  0%|          | 13/5400 [00:00<00:42, 126.58it/s]
moving testing files..
100%|██████████| 5400/5400 [00:38<00:00, 139.22it/s]
```

[7]:

```
# Create a ImageDataGenerator Instance which can be used for data augmentation
train_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=60,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    validation_split=0.3
)

train_generator = train_gen.flow_from_directory(
    directory=TRAIN_DIR,
    target_size=(64, 64),
    batch_size=BATCH_SIZE,
    class_mode=CLASS_MODE,
    subset='training',
    color_mode='rgb',
    shuffle=True,
    seed=69
)
```

```
valid_generator = train_gen.flow_from_directory(
    directory=TRAIN_DIR,
    target_size=(64, 64),
    batch_size=BATCH_SIZE,
    class_mode=CLASS_MODE,
    subset='validation',
    color_mode='rgb',
    shuffle=True,
    seed=69
)

# test generator for evaluation purposes - no augmentations, just rescaling
test_gen = ImageDataGenerator(
    rescale=1./255
)

test_generator = test_gen.flow_from_directory(
    directory=TEST_DIR,
    target_size=(64, 64),
    batch_size=1,
    class_mode=None,
    color_mode='rgb',
    shuffle=False,
    seed=69
)
```

```
Found 15124 images belonging to 10 classes.
Found 6476 images belonging to 10 classes.
Found 5400 images belonging to 10 classes.
```

```
print(train_generator.class_indices)
```

```
{'AnnualCrop': 0, 'Forest': 1, 'HerbaceousVegetation': 2, 'Highway': 3, 'Industrial': 4, 'Pasture': 5, 'PermanentCrop': 6, 'Residential': 7, 'River': 8, 'SeaLake': 9}
```

```
np.save('class_indices', train_generator.class_indices)
```

Training the model

[0]:
```
import tensorflow as tf
from keras.models import Model
from keras.layers import Dense, Dropout, Flatten
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam
from keras.optimizers import Adagrad


from keras.applications import VGG16, VGG19
from keras.applications import ResNet50, ResNet50V2, ResNet152V2
from keras.applications import InceptionV3, Xception

from sklearn.metrics import precision_recall_fscore_support, confusion_matrix, fbeta_score, accuracy_score
```

[11]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
    except RuntimeError as e:
        print(e)

tf.config.set_soft_device_placement(True)
```

```
1 Physical GPUs, 1 Logical GPU
```

Different CNN models

[12]:

```
#for different CNN models we are using different setup of dense layers

def compile_model(cnn_base, input_shape, n_classes, optimizer, fine_tune=None):

    if (cnn_base == 'ResNet50') or (cnn_base == 'ResNet50V2') or (cnn_base == 'ResNet152V2'):
        if cnn_base == 'ResNet50':
            conv_base = ResNet50(include_top=False,
                                 weights='imagenet',
                                 input_shape=input_shape,pooling='avg')
        elif cnn_base == 'ResNet50V2':
            conv_base = ResNet50V2(include_top=False,
                                 weights='imagenet',
                                 input_shape=input_shape,pooling='avg')
        else:
            conv_base = ResNet152V2(include_top=False,
                                 weights='imagenet',
                                 input_shape=input_shape,pooling='avg')
        top_model = conv_base.output
        top_model = Dense(2048, activation='relu')(top_model)


    elif (cnn_base == 'VGG16') or (cnn_base == 'VGG19'):
        if cnn_base == 'VGG16':
            conv_base = VGG16(include_top=False,
                              weights='imagenet',
                              input_shape=input_shape,pooling='avg')
        else:
            conv_base = VGG19(include_top=False,
                              weights='imagenet',
                              input_shape=input_shape,pooling='avg')
        top_model = conv_base.output
        top_model = Dense(2048, activation='relu')(top_model)



    output_layer = Dense(n_classes, activation='softmax')(top_model)
```

```
    model = Model(inputs=conv_base.input, outputs=output_layer)

    if type(fine_tune) == int:
        for layer in conv_base.layers[fine_tune:]:
            layer.trainable = True
    else:
        for layer in conv_base.layers:
            layer.trainable = False

    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
                  metrics=['categorical_accuracy'])

    return model

def plot_history(history):

    acc = history.history['categorical_accuracy']
    val_acc = history.history['val_categorical_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(acc)
    plt.plot(val_acc)
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(loss)
    plt.plot(val_loss)
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')

    plt.show();
```

```
def display_results(y_true, y_preds, class_labels):

    results = pd.DataFrame(precision_recall_fscore_support(y_true, y_preds),
                           columns=class_labels).T
    results.rename(columns={0: 'Precision',
                            1: 'Recall',
                            2: 'F-Score',
                            3: 'Support'}, inplace=True)

    conf_mat = pd.DataFrame(confusion_matrix(y_true, y_preds),
                            columns=class_labels,
                            index=class_labels)
    f2 = fbeta_score(y_true, y_preds, beta=2, average='micro')
    accuracy = accuracy_score(y_true, y_preds)
    print(f"Accuracy: {accuracy}")
    print(f"Global F2 Score: {f2}")
    return results, conf_mat

def plot_predictions(y_true, y_preds, test_generator, class_indices):

    fig = plt.figure(figsize=(20, 10))
    for i, idx in enumerate(np.random.choice(test_generator.samples, size=20, replace=False)):
        ax = fig.add_subplot(4, 5, i + 1, xticks=[], yticks=[])
        ax.imshow(np.squeeze(test_generator[idx]))
        pred_idx = np.argmax(y_preds[idx])
        true_idx = y_true[idx]

        plt.tight_layout()
        ax.set_title("{}\n({})".format(class_indices[pred_idx], class_indices[true_idx]),
                     color=("green" if pred_idx == true_idx else "red"))
```

VGG16

```
vgg16_model = compile_model('VGG16', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2), fine_tune=None)
vgg16_model.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 2s 0us/step
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              1050624
_________________________________________________________________
dense_2 (Dense)              (None, 10)                20490
=================================================================
Total params: 15,785,802
Trainable params: 1,071,114
Non-trainable params: 14,714,688
```

[14]:

```
train_generator.reset()
valid_generator.reset()
N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = valid_generator.samples//BATCH_SIZE
N_EPOCHS = 100

    # model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                             monitor='val_categorical_accuracy',
                             save_best_only=True,
                             verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')
```

```
vgg16_history = vgg16_model.fit_generator(train_generator,
                                steps_per_epoch=N_STEPS,
                                epochs=15,
                                callbacks=[early_stop, checkpoint],
                                validation_data=valid_generator,
                                validation_steps=N_VAL_STEPS)
```

```
Epoch 1/15
236/236 [==============================] - 35s 148ms/step - loss: 1.0645 - categorical_accuracy: 0.6321 - val_loss: 0.8465 - val_categorical_accuracy: 0.6921

Epoch 00001: val_categorical_accuracy improved from -inf to 0.69214, saving model to ../working/model.weights.best.hdf5
Epoch 2/15
236/236 [==============================] - 33s 141ms/step - loss: 0.8035 - categorical_accuracy: 0.7143 - val_loss: 0.8320 - val_categorical_accuracy: 0.7093

Epoch 00002: val_categorical_accuracy improved from 0.69214 to 0.70930, saving model to ../working/model.weights.best.hdf5
Epoch 3/15
236/236 [==============================] - 33s 138ms/step - loss: 0.7446 - categorical_accuracy: 0.7394 - val_loss: 0.6745 - val_categorical_accuracy: 0.7584

Epoch 00003: val_categorical_accuracy improved from 0.70930 to 0.75842, saving model to ../working/model.weights.best.hdf5
Epoch 4/15
236/236 [==============================] - 33s 139ms/step - loss: 0.7252 - categorical_accuracy: 0.7450 - val_loss: 1.0287 - val_categorical_accuracy: 0.7371
```

VGG19

[20]:

```
vgg19_model = compile_model('VGG19', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2), fine_tune=None)
vgg19_model.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg19_weights_tf_dim_
80142336/80134624 [==============================] - 3s 0us/step
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 21,095,498
Trainable params: 1,071,114
Non-trainable params: 20,024,384
_________________________________________________________________
```

[21]:

```
train_generator.reset()
valid_generator.reset()
N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = valid_generator.samples//BATCH_SIZE
N_EPOCHS = 100

    # model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                             monitor='val_categorical_accuracy',
                             save_best_only=True,
                             verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')
```

[22]:

```
vgg19_history = vgg19_model.fit_generator(train_generator,
                          steps_per_epoch=N_STEPS,
                          epochs=15,
                          callbacks=[early_stop, checkpoint],
                          validation_data=valid_generator,
                          validation_steps=N_VAL_STEPS)
```

```
Epoch 1/15
236/236 [==============================] - 36s 154ms/step - loss: 1.3042 - categorical_accuracy: 0.5683 - val_loss: 0.8314 - val_categorical_accuracy: 0.6626

Epoch 00001: val_categorical_accuracy improved from -inf to 0.66259, saving model to ../working/model.weights.best.hdf5
Epoch 2/15
236/236 [==============================] - 36s 151ms/step - loss: 0.9445 - categorical_accuracy: 0.6610 - val_loss: 0.8525 - val_categorical_accuracy: 0.6875

Epoch 00002: val_categorical_accuracy improved from 0.66259 to 0.68746, saving model to ../working/model.weights.best.hdf5
Epoch 3/15
236/236 [==============================] - 35s 148ms/step - loss: 0.8997 - categorical_accuracy: 0.6801 - val_loss: 0.7607 - val_categorical_accuracy: 0.6845

Epoch 00003: val_categorical_accuracy did not improve from 0.68746
Epoch 4/15
236/236 [==============================] - 35s 150ms/step - loss: 0.8611 - categorical_accuracy: 0.6870 - val_loss: 1.1275 - val_categorical_accuracy: 0.6931

Epoch 00004: val_categorical_accuracy improved from 0.68746 to 0.69308, saving model to ../working/model.weights.best.hdf5
Epoch 5/15
236/236 [==============================] - 35s 148ms/step - loss: 0.8289 - categorical_accuracy: 0.7024 - val_loss: 0.5596 - val_categorical_accuracy: 0.7006

Epoch 00005: val_categorical_accuracy improved from 0.69308 to 0.70056, saving model to ../working/model.weights.best.hdf5
Epoch 6/15
236/236 [==============================] - 34s 143ms/step - loss: 0.8126 - categorical_accuracy: 0.7100 - val_loss: 0.6484 - val_categorical_accuracy: 0.7149

Epoch 00006: val_categorical_accuracy improved from 0.70056 to 0.71491, saving model to ../working/model.weights.best.hdf5
Epoch 7/15
236/236 [==============================] - 33s 141ms/step - loss: 0.8139 - categorical_accuracy: 0.7090 - val_loss: 0.8261 - val_categorical_accuracy: 0.7087

Epoch 00007: val_categorical_accuracy did not improve from 0.71491
Epoch 8/15
236/236 [==============================] - 33s 140ms/step - loss: 0.7866 - categorical_accuracy: 0.7195 - val_loss: 0.8437 - val_categorical_accuracy: 0.7283

Epoch 00008: val_categorical_accuracy improved from 0.71491 to 0.72832, saving model to ../working/model.weights.best.hdf5
Epoch 9/15
236/236 [==============================] - 33s 141ms/step - loss: 0.7890 - categorical_accuracy: 0.7212 - val_loss: 1.1410 - val_categorical_accuracy: 0.6366
```

RESNET50

[28]:
```
resnet50_model = compile_model('ResNet50', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2), fine_tune=None)
resnet50_model.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94658560/94653016 [==============================] - 3s 0us/step
Model: "model_3"
```

```
94658560/94653016 [==============================] - 3s 0us/step
Model: "model_3"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 64, 64, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 70, 70, 3)	0	input_3[0][0]
conv1 (Conv2D)	(None, 32, 32, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 32, 32, 64)	256	conv1[0][0]
activation_1 (Activation)	(None, 32, 32, 64)	0	bn_conv1[0][0]
pool1_pad (ZeroPadding2D)	(None, 34, 34, 64)	0	activation_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0	pool1_pad[0][0]
res2a_branch2a (Conv2D)	(None, 16, 16, 64)	4160	max_pooling2d_1[0][0]
bn2a_branch2a (BatchNormalizati	(None, 16, 16, 64)	256	res2a_branch2a[0][0]
activation_2 (Activation)	(None, 16, 16, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 16, 16, 64)	36928	activation_2[0][0]
bn2a_branch2b (BatchNormalizati	(None, 16, 16, 64)	256	res2a_branch2b[0][0]
activation_3 (Activation)	(None, 16, 16, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 16, 16, 256)	16640	activation_3[0][0]
res2a_branch1 (Conv2D)	(None, 16, 16, 256)	16640	max_pooling2d_1[0][0]
bn2a_branch2c (BatchNormalizati	(None, 16, 16, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 16, 16, 256)	1024	res2a_branch1[0][0]
add_1 (Add)	(None, 16, 16, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_4 (Activation)	(None, 16, 16, 256)	0	add_1[0][0]
res2b_branch2a (Conv2D)	(None, 16, 16, 64)	16448	activation_4[0][0]
bn2b_branch2a (BatchNormalizati	(None, 16, 16, 64)	256	res2b_branch2a[0][0]
activation_5 (Activation)	(None, 16, 16, 64)	0	bn2b_branch2a[0][0]

```
Total params: 27,804,554
Trainable params: 4,216,842
Non-trainable params: 23,587,712
```

[35]:

```
train_generator.reset()
valid_generator.reset()
N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = valid_generator.samples//BATCH_SIZE
N_EPOCHS = 100

    # model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                         monitor='val_categorical_accuracy',
                         save_best_only=True,
                         verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5,
                              patience=3, min_lr=0.00001)
```

[36]:

```
resnet50_history = resnet50_model.fit_generator(train_generator,
                           steps_per_epoch=N_STEPS,
                           epochs=25,
                           callbacks=[early_stop, checkpoint,reduce_lr],
                           validation_data=valid_generator,
                           validation_steps=N_VAL_STEPS)
```

```
Epoch 1/25
236/236 [==============================] - 35s 149ms/step - loss: 0.5606 - categorical_accuracy: 0.8394 - val_loss: 53.5593 - val_categorical_accuracy: 0.1129

Epoch 00001: val_categorical_accuracy improved from -inf to 0.11293, saving model to ../working/model.weights.best.hdf5
Epoch 2/25
236/236 [==============================] - 35s 149ms/step - loss: 0.5759 - categorical_accuracy: 0.8345 - val_loss: 49.6126 - val_categorical_accuracy: 0.1128

Epoch 00002: val_categorical_accuracy did not improve from 0.11293
Epoch 3/25
236/236 [==============================] - 35s 147ms/step - loss: 0.5587 - categorical_accuracy: 0.8386 - val_loss: 47.3015 - val_categorical_accuracy: 0.1101

Epoch 00003: val_categorical_accuracy did not improve from 0.11293
Epoch 4/25
236/236 [==============================] - 35s 149ms/step - loss: 0.5714 - categorical_accuracy: 0.8401 - val_loss: 50.8888 - val_categorical_accuracy: 0.1118

Epoch 00004: val_categorical_accuracy did not improve from 0.11293
Epoch 5/25
236/236 [==============================] - 35s 147ms/step - loss: 0.4937 - categorical_accuracy: 0.8570 - val_loss: 59.9598 - val_categorical_accuracy: 0.1135

Epoch 00005: val_categorical_accuracy improved from 0.11293 to 0.11354, saving model to ../working/model.weights.best.hdf5
Epoch 6/25
236/236 [==============================] - 34s 146ms/step - loss: 0.4779 - categorical_accuracy: 0.8607 - val_loss: 56.6331 - val_categorical_accuracy: 0.1134

Epoch 00006: val_categorical_accuracy did not improve from 0.11354
Epoch 7/25
236/236 [==============================] - 35s 148ms/step - loss: 0.4773 - categorical_accuracy: 0.8532 - val_loss: 52.7381 - val_categorical_accuracy: 0.1118

Epoch 00007: val_categorical_accuracy did not improve from 0.11354
Epoch 8/25
236/236 [==============================] - 34s 146ms/step - loss: 0.4671 - categorical_accuracy: 0.8517 - val_loss: 43.8579 - val_categorical_accuracy: 0.1128
```

RESNET50V2

```
resnet50V2_model = compile_model('ResNet50V2', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2), fine_tune=None)
resnet50V2_model.summary()
```

```
Downloading data from https://github.com/keras-team/keras-applications/releases/download/resnet/resnet50v2_weights_tf_dim_orde
94674944/94668760 [==============================] - 4s 0us/step
Model: "model_4"
```

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	(None, 64, 64, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 70, 70, 3)	0	input_4[0][0]
conv1_conv (Conv2D)	(None, 32, 32, 64)	9472	conv1_pad[0][0]
pool1_pad (ZeroPadding2D)	(None, 34, 34, 64)	0	conv1_conv[0][0]
pool1_pool (MaxPooling2D)	(None, 16, 16, 64)	0	pool1_pad[0][0]
conv2_block1_preact_bn (BatchNo	(None, 16, 16, 64)	256	pool1_pool[0][0]
conv2_block1_preact_relu (Activ	(None, 16, 16, 64)	0	conv2_block1_preact_bn[0][0]
conv2_block1_1_conv (Conv2D)	(None, 16, 16, 64)	4096	conv2_block1_preact_relu[0][0]
conv2_block1_1_bn (BatchNormali	(None, 16, 16, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 16, 16, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_pad (ZeroPadding	(None, 18, 18, 64)	0	conv2_block1_1_relu[0][0]
conv2_block1_2_conv (Conv2D)	(None, 16, 16, 64)	36864	conv2_block1_2_pad[0][0]
conv2_block1_2_bn (BatchNormali	(None, 16, 16, 64)	256	conv2_block1_2_conv[0][0]
conv2_block1_2_relu (Activation	(None, 16, 16, 64)	0	conv2_block1_2_bn[0][0]
conv2_block1_0_conv (Conv2D)	(None, 16, 16, 256)	16640	conv2_block1_preact_relu[0][0]
conv2_block1_3_conv (Conv2D)	(None, 16, 16, 256)	16640	conv2_block1_2_relu[0][0]
conv2_block1_out (Add)	(None, 16, 16, 256)	0	conv2_block1_0_conv[0][0] conv2_block1_3_conv[0][0]
conv2_block2_preact_bn (BatchNo	(None, 16, 16, 256)	1024	conv2_block1_out[0][0]
conv2_block2_preact_relu (Activ	(None, 16, 16, 256)	0	conv2_block2_preact_bn[0][0]
conv2_block2_1_conv (Conv2D)	(None, 16, 16, 64)	16384	conv2_block2_preact_relu[0][0]

```
=================================================================================================
Total params: 27,781,642
Trainable params: 4,216,842
Non-trainable params: 23,564,800
_________________________________________________________________________________________________
```

42]:

```
train_generator.reset()
valid_generator.reset()
N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = valid_generator.samples//BATCH_SIZE
N_EPOCHS = 100

    # model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                             monitor='val_categorical_accuracy',
                             save_best_only=True,
                             verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')
```

```
resnet50V2_history = resnet50V2_model.fit_generator(train_generator,
                           steps_per_epoch=N_STEPS,
                           epochs=15,
                           callbacks=[early_stop, checkpoint],
                           validation_data=valid_generator,
                           validation_steps=N_VAL_STEPS)
```

```
Epoch 1/15
236/236 [==============================] - 34s 145ms/step - loss: 1.1983 - categorical_accuracy: 0.6274 - val_loss: 1.6402 - val_categorical_accuracy: 0.4592

Epoch 00001: val_categorical_accuracy improved from -inf to 0.45916, saving model to ../working/model.weights.best.hdf5
Epoch 2/15
236/236 [==============================] - 35s 147ms/step - loss: 1.1302 - categorical_accuracy: 0.6511 - val_loss: 1.4655 - val_categorical_accuracy: 0.4301

Epoch 00002: val_categorical_accuracy did not improve from 0.45916
Epoch 3/15
236/236 [==============================] - 34s 146ms/step - loss: 1.0766 - categorical_accuracy: 0.6627 - val_loss: 1.5598 - val_categorical_accuracy: 0.4593

Epoch 00003: val_categorical_accuracy improved from 0.45916 to 0.45930, saving model to ../working/model.weights.best.hdf5
Epoch 4/15
236/236 [==============================] - 35s 147ms/step - loss: 1.0619 - categorical_accuracy: 0.6728 - val_loss: 1.3509 - val_categorical_accuracy: 0.4588

Epoch 00004: val_categorical_accuracy did not improve from 0.45930
Epoch 5/15
236/236 [==============================] - 34s 146ms/step - loss: 1.0291 - categorical_accuracy: 0.6800 - val_loss: 1.6435 - val_categorical_accuracy: 0.4502

Epoch 00005: val_categorical_accuracy did not improve from 0.45930
Epoch 6/15
236/236 [==============================] - 35s 147ms/step - loss: 1.0399 - categorical_accuracy: 0.6772 - val_loss: 1.1262 - val_categorical_accuracy: 0.5080

Epoch 00006: val_categorical_accuracy improved from 0.45930 to 0.50795, saving model to ../working/model.weights.best.hdf5
Epoch 7/15
236/236 [==============================] - 34s 144ms/step - loss: 1.0153 - categorical_accuracy: 0.6846 - val_loss: 1.4200 - val_categorical_accuracy: 0.4800

Epoch 00007: val_categorical_accuracy did not improve from 0.50795
Epoch 8/15
236/236 [==============================] - 34s 144ms/step - loss: 1.0085 - categorical_accuracy: 0.6932 - val_loss: 1.4243 - val_categorical_accuracy: 0.4574

Epoch 00008: val_categorical_accuracy did not improve from 0.50795
Epoch 9/15
236/236 [==============================] - 34s 144ms/step - loss: 1.0126 - categorical_accuracy: 0.6903 - val_loss: 1.8554 - val_categorical_accuracy: 0.4470
```