Neural Network Object Detection

using Caltech 256 object dataset

In [1]:
```
import os
import datetime
import numpy

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv2D, BatchNormalization, Activation
from tensorflow.keras.layers import AveragePooling2D, Input, Flatten
from tensorflow.keras import activations
from tensorflow.keras.regularizers import l2

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import os
import zipfile
import random
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from shutil import copyfile
from tensorflow import keras
from tensorflow.keras import layers
```

In [2]:
```
train_datagen = ImageDataGenerator(rescale=1.0/255.)
train_generator = train_datagen.flow_from_directory('256_data/256_data/training/',
                                                    batch_size=50,

                                                    target_size=(150, 150))

test_datagen = ImageDataGenerator(rescale=1.0/255.)
test_generator = test_datagen.flow_from_directory('256_data/256_data/testing/',
                                                    batch_size=50,

                                                    target_size=(150, 150))
```

```
Found 21308 images belonging to 257 classes.
Found 9299 images belonging to 257 classes.
```

In [3]:
```python
def resnet_layer(inputs,
                 num_filters=16,
                 kernel_size=3,
                 strides=1,
                 activation='relu',
                 batch_normalization=True,
                 conv_first=True):
    """2D Convolution-Batch Normalization-Activation stack builder

    # Arguments
        inputs (tensor): input tensor from input image or previous layer
        num_filters (int): Conv2D number of filters
        kernel_size (int): Conv2D square kernel dimensions
        strides (int): Conv2D square stride dimensions
        activation (string): activation name
        batch_normalization (bool): whether to include batch normalization
        conv_first (bool): conv-bn-activation (True) or
            bn-activation-conv (False)

    # Returns
        x (tensor): tensor as input to the next layer
    """
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))

    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x


def resnet_v1(inputs, filters, num_res_blocks, pool_size):
    """ResNet Version 1 Model builder [a]

    Stacks of 2 x (3 x 3) Conv2D-BN-ReLU
    Last ReLU is after the shortcut connection.
    At the beginning of each stage, the feature map size is halved (downsampled)
    by a convolutional layer with strides=2, while the number of filters is
    doubled. Within each stage, the layers have the same number filters and the
    same number of filters.

    # Arguments
        inputs (layer):         the input tensor
        filters ([int]):        number of filters in each stage, length of list determines number of stages
        num_res_blocks (int):   number of residual blocks per stage
        pool_size (int):        size of the average pooling at the end

    # Returns
        output after global average pooling and flatten, ready for output
    """
    x = resnet_layer(inputs=inputs,
                     num_filters=filters[0])

    # Instantiate the stack of residual units
    for stack, filters in enumerate(filters):
        for res_block in range(num_res_blocks):
            strides = 1
            if stack > 0 and res_block == 0:  # first layer but not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=filters,
                             activation=None)
            if stack > 0 and res_block == 0:  # first layer but not first stack
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)

    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=pool_size)(x)
    y = Flatten()(x)

    return y


def resnet_v2(inputs, filters, num_res_blocks, pool_size):
    """ResNet Version 2 Model builder [b]
```

```
    Stacks of (1 x 1)-(3 x 3)-(1 x 1) BN-ReLU-Conv2D or also known as
    bottleneck layer
    First shortcut connection per layer is 1 x 1 Conv2D.
    Second and onwards shortcut connection is identity.
    At the beginning of each stage, the feature map size is halved (downsampled)
    by a convolutional layer with strides=2, while the number of filter maps is
    doubled. Within each stage, the layers have the same number filters and the
    same filter map sizes.

    # Arguments
        inputs (layer):         the input tensor
        filters ([int]):        number of filters in each stage, length of list determines number of stages
        num_res_blocks (int):   number of residual blocks per stage
        pool_size (int):        size of the average pooling at the end

    # Returns
        output after global average pooling and flatten, ready for output
    """

    x = resnet_layer(inputs=inputs,
                     num_filters=filters[0],
                     conv_first=True)

    # Instantiate the stack of residual units
    for stage, filters in enumerate(filters):
        num_filters_in = filters
        for res_block in range(num_res_blocks):
            activation = 'relu'
            batch_normalization = True
            strides = 1
            if stage == 0:
                num_filters_out = num_filters_in * 4
                if res_block == 0:  # first layer and first stage
                    activation = None
                    batch_normalization = False
            else:
                num_filters_out = num_filters_in * 2
                if res_block == 0:  # first layer but not first stage
                    strides = 2    # downsample

            # bottleneck residual unit
            y = resnet_layer(inputs=x,
                             num_filters=num_filters_in,
                             kernel_size=1,
                             strides=strides,
                             activation=activation,
                             batch_normalization=batch_normalization,
                             conv_first=False)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters_in,
                             conv_first=False)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters_out,
                             kernel_size=1,
                             conv_first=False)
            if res_block == 0:
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters_out,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])

        num_filters_in = num_filters_out

    # Add classifier on top.
    # v2 has BN-ReLU before Pooling
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = AveragePooling2D(pool_size=pool_size)(x)
    y = Flatten()(x)
    return y
```

In [4]:

```
gpus= tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)
```

In [6]:
```
inputs = keras.Input(shape=(150, 150, 3, ), name='img')

x = resnet_v1(inputs, [16, 32], 1, 14)
outputs = keras.layers.Dense(257, activation = 'softmax')(x)

model_resnet_v1 = keras.Model(inputs=inputs, outputs=outputs, name='simple_resnet_v1')
model_resnet_v1.summary()
```

```
Model: "simple_resnet_v1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
img (InputLayer)                [(None, 150, 150, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 150, 150, 16) 448         img[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 150, 150, 16) 64          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 150, 150, 16) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 150, 150, 16) 2320        activation[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 150, 150, 16) 64          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 150, 150, 16) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 150, 150, 16) 2320        activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 150, 150, 16) 64          conv2d_2[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 150, 150, 16) 0           activation[0][0]
                                                                 batch_normalization_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 150, 150, 16) 0           add[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 75, 75, 32)   4640        activation_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 75, 75, 32)   128         conv2d_3[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 75, 75, 32)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 75, 75, 32)   9248        activation_3[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 75, 75, 32)   544         activation_2[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 75, 75, 32)   128         conv2d_4[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 75, 75, 32)   0           conv2d_5[0][0]
                                                                 batch_normalization_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 75, 75, 32)   0           add_1[0][0]
__________________________________________________________________________________________________
average_pooling2d (AveragePooli (None, 5, 5, 32)     0           activation_4[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 800)          0           average_pooling2d[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 257)          205857      flatten[0][0]
==================================================================================================
Total params: 225,825
Trainable params: 225,601
Non-trainable params: 224
__________________________________________________________________________________________________
```

In [7]:
```
model_resnet_v1.compile(loss=keras.losses.CategoricalCrossentropy(),
                  optimizer=keras.optimizers.RMSprop(),
                  metrics=['accuracy'])
history = model_resnet_v1.fit(train_generator,
                        epochs=50,
                        validation_data = test_generator)
```

```
Epoch 1/50
427/427 [==============================] - 168s 373ms/step - loss: 4.8038 - accuracy: 0.1245 - val_loss: 4.4447 - val_accuracy: 0.1692
Epoch 2/50
427/427 [==============================] - 33s 77ms/step - loss: 3.9143 - accuracy: 0.2315 - val_loss: 4.1503 - val_accuracy: 0.2042
Epoch 3/50
427/427 [==============================] - 33s 77ms/step - loss: 3.5148 - accuracy: 0.2917 - val_loss: 3.8474 - val_accuracy: 0.2505
Epoch 4/50
427/427 [==============================] - 33s 77ms/step - loss: 3.2352 - accuracy: 0.3348 - val_loss: 3.8919 - val_accuracy: 0.2471
Epoch 5/50
427/427 [==============================] - 33s 77ms/step - loss: 3.0178 - accuracy: 0.3699 - val_loss: 3.7823 - val_accuracy: 0.2798
Epoch 6/50
427/427 [==============================] - 33s 77ms/step - loss: 2.8349 - accuracy: 0.4009 - val_loss: 3.8480 - val_accuracy: 0.2765
Epoch 7/50
427/427 [==============================] - 33s 77ms/step - loss: 2.6833 - accuracy: 0.4305 - val_loss: 3.7142 - val_accuracy: 0.2918
Epoch 8/50
427/427 [==============================] - 33s 77ms/step - loss: 2.5507 - accuracy: 0.4535 - val_loss: 3.7178 - val_accuracy: 0.2916
Epoch 9/50
427/427 [==============================] - 33s 77ms/step - loss: 2.4338 - accuracy: 0.4725 - val_loss: 3.4961 - val_accuracy: 0.3167
Epoch 10/50
427/427 [==============================] - 33s 77ms/step - loss: 2.3385 - accuracy: 0.4919 - val_loss: 3.6728 - val_accuracy: 0.3170
Epoch 11/50
427/427 [==============================] - 33s 77ms/step - loss: 2.2533 - accuracy: 0.5076 - val_loss: 3.8985 - val_accuracy: 0.3132
Epoch 12/50
427/427 [==============================] - 33s 77ms/step - loss: 2.1666 - accuracy: 0.5210 - val_loss: 3.5732 - val_accuracy: 0.3201
Epoch 13/50
427/427 [==============================] - 34s 79ms/step - loss: 2.0984 - accuracy: 0.5346 - val_loss: 3.6465 - val_accuracy: 0.3309
Epoch 14/50
427/427 [==============================] - 33s 76ms/step - loss: 2.0314 - accuracy: 0.5507 - val_loss: 3.5605 - val_accuracy: 0.3242
Epoch 15/50
427/427 [==============================] - 33s 77ms/step - loss: 1.9705 - accuracy: 0.5588 - val_loss: 3.8112 - val_accuracy: 0.3152
Epoch 16/50
427/427 [==============================] - 33s 76ms/step - loss: 1.9068 - accuracy: 0.5704 - val_loss: 3.6865 - val_accuracy: 0.3270
Epoch 17/50
427/427 [==============================] - 33s 77ms/step - loss: 1.8582 - accuracy: 0.5815 - val_loss: 3.6308 - val_accuracy: 0.3392
Epoch 18/50
427/427 [==============================] - 33s 78ms/step - loss: 1.7980 - accuracy: 0.5919 - val_loss: 4.0615 - val_accuracy: 0.3090
Epoch 19/50
427/427 [==============================] - 32s 76ms/step - loss: 1.7512 - accuracy: 0.6030 - val_loss: 3.7672 - val_accuracy: 0.3379
Epoch 20/50
427/427 [==============================] - 33s 77ms/step - loss: 1.7008 - accuracy: 0.6125 - val_loss: 3.8987 - val_accuracy: 0.3339
Epoch 21/50
427/427 [==============================] - 33s 77ms/step - loss: 1.6582 - accuracy: 0.6200 - val_loss: 4.7086 - val_accuracy: 0.2426
Epoch 22/50
427/427 [==============================] - 33s 76ms/step - loss: 1.6144 - accuracy: 0.6288 - val_loss: 3.9051 - val_accuracy: 0.3253
Epoch 23/50
427/427 [==============================] - 33s 76ms/step - loss: 1.5682 - accuracy: 0.6411 - val_loss: 4.2705 - val_accuracy: 0.2813
Epoch 24/50
427/427 [==============================] - 33s 76ms/step - loss: 1.5291 - accuracy: 0.6451 - val_loss: 3.9966 - val_accuracy: 0.3290
Epoch 25/50
427/427 [==============================] - 33s 77ms/step - loss: 1.4925 - accuracy: 0.6567 - val_loss: 4.0347 - val_accuracy: 0.3219
Epoch 26/50
427/427 [==============================] - 33s 78ms/step - loss: 1.4567 - accuracy: 0.6638 - val_loss: 3.9781 - val_accuracy: 0.3297
Epoch 27/50
427/427 [==============================] - 33s 77ms/step - loss: 1.4166 - accuracy: 0.6689 - val_loss: 4.1706 - val_accuracy: 0.2987
Epoch 28/50
427/427 [==============================] - 33s 78ms/step - loss: 1.3889 - accuracy: 0.6741 - val_loss: 4.6050 - val_accuracy: 0.3177
Epoch 29/50
427/427 [==============================] - 33s 77ms/step - loss: 1.3513 - accuracy: 0.6846 - val_loss: 4.1143 - val_accuracy: 0.3168
Epoch 30/50
427/427 [==============================] - 33s 77ms/step - loss: 1.3277 - accuracy: 0.6864 - val_loss: 3.9754 - val_accuracy: 0.3324
Epoch 31/50
427/427 [==============================] - 33s 78ms/step - loss: 1.2949 - accuracy: 0.6961 - val_loss: 4.7213 - val_accuracy: 0.2777
Epoch 32/50
427/427 [==============================] - 33s 77ms/step - loss: 1.2637 - accuracy: 0.7037 - val_loss: 4.3123 - val_accuracy: 0.3187
Epoch 33/50
427/427 [==============================] - 33s 76ms/step - loss: 1.2328 - accuracy: 0.7084 - val_loss: 4.2692 - val_accuracy: 0.3226
Epoch 34/50
427/427 [==============================] - 33s 78ms/step - loss: 1.2038 - accuracy: 0.7147 - val_loss: 4.5010 - val_accuracy: 0.3296
Epoch 35/50
427/427 [==============================] - 32s 76ms/step - loss: 1.1742 - accuracy: 0.7196 - val_loss: 4.4372 - val_accuracy: 0.3226
Epoch 36/50
427/427 [==============================] - 33s 77ms/step - loss: 1.1546 - accuracy: 0.7263 - val_loss: 4.5646 - val_accuracy: 0.3250
Epoch 37/50
427/427 [==============================] - 33s 78ms/step - loss: 1.1246 - accuracy: 0.7293 - val_loss: 4.6323 - val_accuracy: 0.3053
Epoch 38/50
427/427 [==============================] - 33s 77ms/step - loss: 1.1094 - accuracy: 0.7361 - val_loss: 4.4618 - val_accuracy: 0.3259
Epoch 39/50
427/427 [==============================] - 33s 76ms/step - loss: 1.0773 - accuracy: 0.7428 - val_loss: 4.5576 - val_accuracy: 0.3240
Epoch 40/50
427/427 [==============================] - 32s 76ms/step - loss: 1.0494 - accuracy: 0.7490 - val_loss: 4.5706 - val_accuracy: 0.3278
Epoch 41/50
427/427 [==============================] - 32s 76ms/step - loss: 1.0381 - accuracy: 0.7534 - val_loss: 4.8351 - val_accuracy: 0.3114
Epoch 42/50
427/427 [==============================] - 32s 76ms/step - loss: 1.0127 - accuracy: 0.7531 - val_loss: 4.5224 - val_accuracy: 0.3243
Epoch 43/50
427/427 [==============================] - 33s 76ms/step - loss: 0.9976 - accuracy: 0.7599 - val_loss: 4.8681 - val_accuracy: 0.3206
Epoch 44/50
427/427 [==============================] - 33s 77ms/step - loss: 0.9765 - accuracy: 0.7629 - val_loss: 4.7577 - val_accuracy: 0.3229
Epoch 45/50
427/427 [==============================] - 33s 78ms/step - loss: 0.9485 - accuracy: 0.7714 - val_loss: 4.5852 - val_accuracy: 0.3301
Epoch 46/50
427/427 [==============================] - 32s 76ms/step - loss: 0.9369 - accuracy: 0.7735 - val_loss: 4.9665 - val_accuracy: 0.3209
Epoch 47/50
427/427 [==============================] - 33s 76ms/step - loss: 0.9102 - accuracy: 0.7788 - val_loss: 5.0101 - val_accuracy: 0.2993
Epoch 48/50
427/427 [==============================] - 33s 77ms/step - loss: 0.9014 - accuracy: 0.7811 - val_loss: 4.8024 - val_accuracy: 0.3180
Epoch 49/50
427/427 [==============================] - 32s 76ms/step - loss: 0.8884 - accuracy: 0.7824 - val_loss: 5.0152 - val_accuracy: 0.3055
Epoch 50/50
427/427 [==============================] - 33s 77ms/step - loss: 0.8651 - accuracy: 0.7904 - val_loss: 5.0373 - val_accuracy: 0.3137
```