# Topics: Data augmentation for imported data



In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [20]:
```
import os
import tensorflow as tf
import numpy as np

SEED = 1234
tf.random.set_seed(SEED)

# Get current working directory
cwd = os.getcwd()
```

We want to train the network on the dataset by loading the images at runtime (since we cannot have them all in memory)

In [7]:
```
# Run this cell only if you are using Colab with Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [9]:
```
# Run this cell only if you are using Colab with Drive
!unzip '/content/drive/My Drive/UCMerced_LandUse.zip'
```

In [16]:
```
!ls /content                    # just to check what we have in 'content'
!ls /content/UCMerced_LandUse/  # let's inspect the dataset: we have test, training, validation
                                # (usually we have to do this division manually)
```

```
drive  sample_data  UCMerced_LandUse
test  training  validation
```

In [17]:
```
!ls /content/UCMerced_LandUse/training
```

```
agricultural      denseresidential  mediumresidential  sparseresidential
airplane          forest            mobilehomepark     storagetanks
baseballdiamond   freeway           overpass           tenniscourt
beach             golfcourse        parkinglot
buildings         harbor            river
chaparral         intersection      runway
```

# Example: Multi-class Land Use classification

## Uc Merced Land Use Dataset (21 classes, 100 images each)

### Directory structure

this must be the structure of our data if we want the network to understand all itself

If we give the right structure to the data then flow_from_directory(..., class_mode='categorical', ..) will recognize the structure

```
- UCMerced_LandUse/
    - training/
        - agricultural/
            - img1, img2, …, imgN
        - …
        - parkinglot/
            - img1, img2, ... , imgN
    - validation/
        - agricultural/
            - img1, img2, …, imgN
        - …
        - parkinglot/
            - img1, img2, ... , imgN
    - test/
        - agricultural/
            - img1, img2, …, imgN
        - …
        - parkinglot/
            - img1, img2, ... , imgN
```

we apply some data augmentation through ImageDataGenerator(..)

In [19]:
```
# ImageDataGenerator
# ------------------------------------------------------------------------------
from tensorflow.keras.preprocessing.image import ImageDataGenerator

apply_data_augmentation = True

# Create training ImageDataGenerator object
if apply_data_augmentation:
    train_data_gen = ImageDataGenerator(rotation_range=10,
                                        width_shift_range=10,
                                        height_shift_range=10,
                                        zoom_range=0.3,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        fill_mode='constant',
                                        cval=0,
                                        rescale=1./255)
else:
    train_data_gen = ImageDataGenerator(rescale=1./255)

# Create validation and test ImageDataGenerator objects
valid_data_gen = ImageDataGenerator(rescale=1./255)
test_data_gen = ImageDataGenerator(rescale=1./255)
```

NB. we're just doing random transforms of the images, we're not producing other images to add to the dataset (the dimension of the dataset will not vary)

the image is randomly rotated with a degree randomly (uniformly) sampled between -10 and +10

the transformations are combined among eachother

Same reasoning applies for other transformations

zoom up to -30% and 30% of the image

original → augmented (eg. rotated)

what do we do in ■? we can fill it with a constant established by cval

the rize of each image is multiplied by 1/255 (rescaled)

In [21]:
```
# Create generators to read images from dataset directory
# ------------------------------------------------------------------------------
dataset_dir = os.path.join(cwd, 'UCMerced_LandUse')

bs = 8            # batch size
img_h = 256       # image shape (1/2)
img_w = 256       # image shape (2/2)
num_classes = 21  # number of classes
```

(we've still not "creating" dataset)

we have to say where to find the dataset, the training, the validation, the test

```
# Training
training_dir = os.path.join(dataset_dir, 'training')
train_gen = train_data_gen.flow_from_directory(training_dir,
                                               batch_size=bs,
                                               class_mode='categorical',
                                               shuffle=True,
                                               seed=SEED)  # targets are directly converted into one-hot vectors

# Validation
validation_dir = os.path.join(dataset_dir, 'validation')
valid_gen = valid_data_gen.flow_from_directory(validation_dir,
                                               batch_size=bs,
                                               class_mode='categorical',
                                               shuffle=False,
                                               seed=SEED)

# Test
test_dir = os.path.join(dataset_dir, 'test')
test_gen = test_data_gen.flow_from_directory(test_dir,
                                             batch_size=bs,
                                             class_mode='categorical',
                                             shuffle=False,
                                             seed=SEED)
```

we're going to consider images in batches

this because it's a multiple-class image classification

```
Found 1470 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
```

this informs us that the data has been generated

In [23]:
```
# Create Dataset objects
# ----------------------------------------------------------------------
# Training
# ----------------------------------------------------------------------
train_dataset = tf.data.Dataset.from_generator(lambda: train_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))

# The generator substitutes:
# 1.  train_dataset = train_dataset.shuffle(buffer_size=len(train_gen))
# 2.  def normalize_img(x_, y_):
#         return tf.cast(x_, tf.float32) / 255., y_
#     train_dataset = train_dataset.map(normalize_img)
# 3.  def to_categorical(x_, y_):
#         return x_, tf.one_hot(y_, depth=10)
#     train_dataset = train_dataset.map(to_categorical)
# 4.  train_dataset = train_dataset.batch(bs)

# Repeat
# Without calling the repeat function the dataset will be empty after consuming all the images
# (the training will stop after one epoch)
train_dataset = train_dataset.repeat()

# Validation
# ----------------------------------------------------------------------
valid_dataset = tf.data.Dataset.from_generator(lambda: valid_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))

valid_dataset = valid_dataset.repeat()

# Test
# ----------------------------------------------------------------------
test_dataset = tf.data.Dataset.from_generator(lambda: test_gen,
                                              output_types=(tf.float32, tf.float32),
                                              output_shapes=([None, img_h, img_w, 3], [None, num_classes]))

test_dataset = valid_dataset.repeat()
```

(now we create the dataset)

for images

for classes

| Shuffle
| Normalization of images
| 1-hot encoding
| (for categorical cross-entropy)
| Division in batches

In [24]:
```
# Let's test data augmentation
# ----------------------------------------------------------------------
import time
import matplotlib.pyplot as plt
%matplotlib inline

iterator = iter(train_dataset)
```

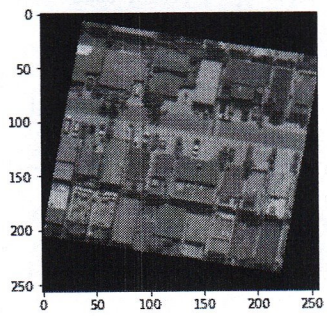
← using this, if we re-run (↓) we get another image

In [32]:
```
augmented_img, target = next(iterator)
augmented_img = np.array(augmented_img[0])   # First element
augmented_img = augmented_img * 255          # denormalize

plt.imshow(np.uint8(augmented_img))
# fig.canvas.draw()
plt.plot()
```

for visualization

Out[32]: <matplotlib.image.AxesImage at 0x7fc8c22dfda0>

Out[32]: []



# Topics 2/3: CNN

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np
SEED = 1234
tf.random.set_seed(SEED)
cwd = os.getcwd()  # get current working directory
```

# Example: Multi-class Land Use classification

## Uc Merced Land Use Dataset

In [3]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

apply_data_augmentation = False

# Create training ImageDataGenerator object
if apply_data_augmentation:
    train_data_gen = ImageDataGenerator(rotation_range=10,
                                        width_shift_range=10,
                                        height_shift_range=10,
                                        zoom_range=0.3,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        fill_mode='constant',
                                        cval=0,
                                        rescale=1./255)
else:
    train_data_gen = ImageDataGenerator(rescale=1./255)

# Create validation and test ImageDataGenerator objects
valid_data_gen = ImageDataGenerator(rescale=1./255)
test_data_gen = ImageDataGenerator(rescale=1./255)
```

In [4]:
```
# Run this cell only if you are using Colab with Drive
from google.colab import drive
drive.mount('/content/drive')
```

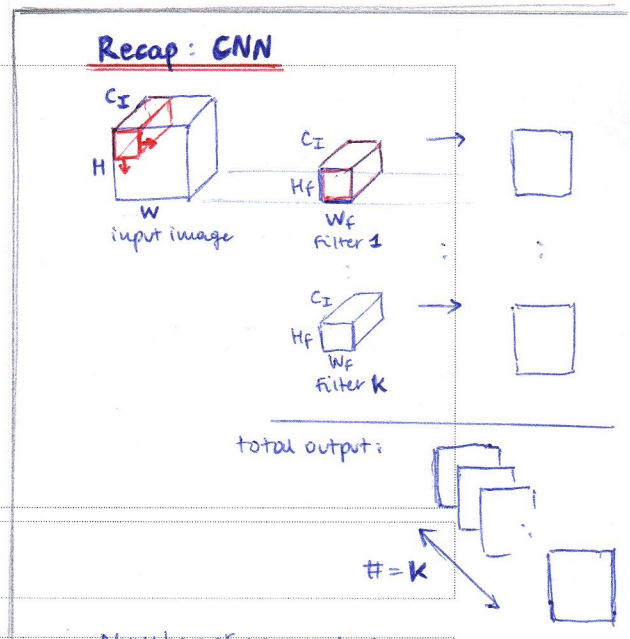
```
Mounted at /content/drive
```

In [5]:
```
# Run this cell only if you are using Colab with Drive
!unzip '/content/drive/My Drive/UCMerced_LandUse.zip'
```

In [6]:
```
!ls /content
```

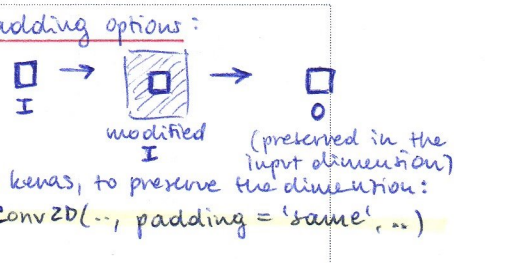
```
drive  sample_data  UCMerced_LandUse
```

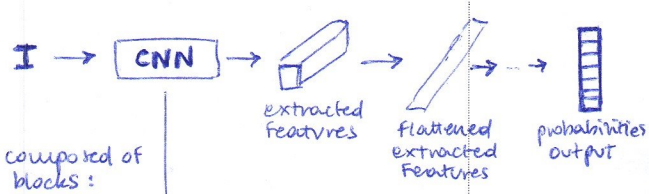
In [7]:
```
# Create generators to read images from dataset directory
# -------------------------------------------------------------------
dataset_dir = os.path.join(cwd, 'UCMerced_LandUse')

bs = 8            # batch size
img_h = 256       # image shape (1/2)
img_w = 256       # image shape (2/2)
num_classes = 21  # number of classes

decide_class_indices = False
if decide_class_indices:
    classes = ['agricultural',       # 0
               'airplane',           # 1
               'baseballdiamond',    # 2
               'beach',              # 3
               'buildings',          # 4
               'chaparral',          # 5
               'denseresidensial',   # 6
               'forest',             # 7
               'freeway',            # 8
               'golfcourse',         # 9
               'harbor',             # 10
               'intersection',       # 11
               'mediumresidential',  # 12
               'mobilehomepark',     # 13
               'overpass',           # 14
               'parkinglot',         # 15
               'river',              # 16
               'runway',             # 17
               'sparseresidential',  # 18
               'storagetanks',       # 19
               'tenniscourt']        # 20
else:
    classes=None

# Training
training_dir = os.path.join(dataset_dir, 'training')
train_gen = train_data_gen.flow_from_directory(training_dir,
                                               batch_size=bs,
                                               classes=classes,
                                               class_mode='categorical',
                                               shuffle=True,
                                               seed=SEED)  # targets are directly converted into one-hot vectors

# Validation
validation_dir = os.path.join(dataset_dir, 'validation')
valid_gen = valid_data_gen.flow_from_directory(validation_dir,
                                               batch_size=bs,
                                               classes=classes,
                                               class_mode='categorical',
```




Number of parameters for every convolution:

$\#params = H_f \times W_f \times C_I \times k$ (k = # filters)




composed of blocks:

each block has the same structure:
- convolution (through filters)
- activation function
- pooling layer

Tipically, as we go deeper (→) we increase the # of filters. Filters recognize patterns; in the first layers we have small receptive field, we can only recognize some local patterns (not interested) (very low level features). As we go deeper we extract features of higher level of abstraction and we can capture more (general) features, that's why we increase the # of filters.

```
                                            shuffle=False,
                                            seed=SEED)

# Test
test_dir = os.path.join(dataset_dir, 'test')
test_gen = test_data_gen.flow_from_directory(test_dir,
                                             batch_size=bs,
                                             classes=classes,
                                             class_mode='categorical',
                                             shuffle=False,
                                             seed=SEED)
```

```
Found 1470 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
```

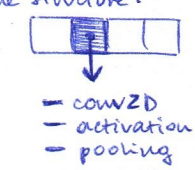
In [8]:
```
# Check how keras assigned the labels
train_gen.class_indices
```

Out[8]:
```
{'agricultural': 0,
 'airplane': 1,
 'baseballdiamond': 2,
 'beach': 3,
 'buildings': 4,
 'chaparral': 5,
 ..
 'river': 16,
 'runway': 17,
 'sparseresidential': 18,
 'storagetanks': 19,
 'tenniscourt': 20}
```

In [9]:
```
# Create Dataset objects
# ------------------------------------------------------------------------------
# Training
train_dataset = tf.data.Dataset.from_generator(lambda: train_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
train_dataset = train_dataset.repeat()

# Validation
valid_dataset = tf.data.Dataset.from_generator(lambda: valid_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
valid_dataset = valid_dataset.repeat()

# Test
test_dataset = tf.data.Dataset.from_generator(lambda: test_gen,
                                              output_types=(tf.float32, tf.float32),
                                              output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
test_dataset = valid_dataset.repeat()
```

In [10]:
```
# Architecture
# ------------------------------------------------------------------------------
# Features extraction
start_f = 8     <- # features extracted from the first convolution
depth   = 5
model   = tf.keras.Sequential()

# Features extraction
for i in range(depth):

    if i == 0:
        input_shape = [img_h, img_w, 3]
    else:
        input_shape=[None]   = "unknown" = "adattati"

    # Conv block: Conv2D -> Activation -> Pooling
    model.add(tf.keras.layers.Conv2D(filters=start_f,   <- # filters
                                     kernel_size=(3, 3),
                                     strides=(1, 1),
                                     padding='same',    <- size(output) = size(input)
                                     input_shape=input_shape))
    model.add(tf.keras.layers.ReLU())
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))   <- in this way we've halfing the spatial dimension

    start_f *= 2   <- at each convolution we increase the number of extracted features (+2)

# Classifier
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=512, activation='relu'))      <- intermediate layer
model.add(tf.keras.layers.Dense(units=num_classes, activation='softmax'))   <- classifier layer

# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

We implement the architecture with a sequential model (since the CNN is composed by blocks that have the same structure:



- conv2D
- activation
- pooling

we can implement it with a for cycle)

```
Model: "sequential"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 256, 256, 8) | 224 |
| re_lu (ReLU) | (None, 256, 256, 8) | 0 |
| max_pooling2d (MaxPooling2D) | (None, 128, 128, 8) | 0 |
| conv2d_1 (Conv2D) | (None, 128, 128, 16) | 1168 |
| re_lu_1 (ReLU) | (None, 128, 128, 16) | 0 |
| max_pooling2d_1 (MaxPooling2 | (None, 64, 64, 16) | 0 |
| conv2d_2 (Conv2D) | (None, 64, 64, 32) | 4640 |

features extractor (pt. 1)

$[H_f \times W_f \times k] \times C_I + \underbrace{k}_{\text{biases}} = [3 \times 3 \times 8] \times 3 + 8$ ("kernel-size")

$[H_f \times W_f \times k] \times C_I + \underbrace{k}_{\text{biases}} = (3 \times 3 \times 16) \times 8 + 16$ ("kernel-size" = filter-size)

feature extractor (pt. 2)

classifier

| | | |
|---|---|---|
| re_lu_2 (ReLU) | (None, 64, 64, 32) | 0 |
| max_pooling2d_2 (MaxPooling2 | (None, 32, 32, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 32, 32, 64) | 18496 |
| re_lu_3 (ReLU) | (None, 32, 32, 64) | 0 |
| max_pooling2d_3 (MaxPooling2 | (None, 16, 16, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 16, 16, 128) | 73856 |
| re_lu_4 (ReLU) | (None, 16, 16, 128) | 0 |
| max_pooling2d_4 (MaxPooling2 | (None, 8, 8, 128) | 0 |
| flatten (Flatten) | (None, 8192) | 0 |
| dense (Dense) | (None, 512) | 4194816 |
| dense_1 (Dense) | (None, 21) | 10773 |

```
Total params: 4,303,973
Trainable params: 4,303,973
Non-trainable params: 0
```

In [20]:

```
# In this cell we create the same model of the cell before, but we implement it
# by inheriting from tf.keras.Model. This is to provide you with an example of how
# we can create customized models. DO NOT run this cell if you have already created
# the model in the cell above.

start_f = 8
depth   = 5

# Create convolutional block
class ConvBlock(tf.keras.Model):
    def __init__(self, num_filters):
        super(ConvBlock, self).__init__()
        self.conv2d = tf.keras.layers.Conv2D(filters=num_filters,
                                             kernel_size=(3, 3),
                                             strides=(1, 1),
                                             padding='same')
        self.activation = tf.keras.layers.ReLU()  # we can specify the activation function directly in Conv2D
        self.pooling = tf.keras.layers.MaxPool2D(pool_size=(2, 2))

    def call(self, inputs):
        x = self.conv2d(inputs)
        x = self.activation(x)
        x = self.pooling(x)
        return x

class CNNClassifier(tf.keras.Model):
    def __init__(self, depth, start_f, num_classes):
        super(CNNClassifier, self).__init__()

        self.feature_extractor = tf.keras.Sequential()

        for i in range(depth):
            self.feature_extractor.add(ConvBlock(num_filters=start_f))
            start_f *= 2

        self.flatten = tf.keras.layers.Flatten()
        self.classifier = tf.keras.Sequential()
        self.classifier.add(tf.keras.layers.Dense(units=512, activation='relu'))
        self.classifier.add(tf.keras.layers.Dense(units=num_classes, activation='softmax'))

    def call(self, inputs):
        x = self.feature_extractor(inputs)
        x = self.flatten(x)
        x = self.classifier(x)
        return x

# Create Model instance
model = CNNClassifier(depth=depth,
                      start_f=start_f,
                      num_classes=num_classes)
# Build Model (Required)
model.build(input_shape=(None, img_h, img_w, 3))

# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

```
Model: "cnn_classifier"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| sequential_1 (Sequential) | (None, 8, 8, 128) | 98384 |
| flatten_1 (Flatten) | multiple | 0 |
| sequential_2 (Sequential) | (None, 21) | 4205589 |

```
Total params: 4,303,973
Trainable params: 4,303,973
Non-trainable params: 0
```

from here it's just a classification task!

In [11]:

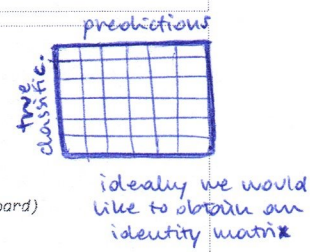
```
# Optimization params
# -------------------------------------------------------------------------------------------
loss      = tf.keras.losses.CategoricalCrossentropy()   # Loss
lr        = 1e-4                                        # Learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)  # Optimizer
```

```
# Validation metrics
# ----------------------------------------------------------------------------------
metrics = ['accuracy']

# Compile Model
# ----------------------------------------------------------------------------------
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

In [17]:
```
%load_ext tensorboard
%tensorboard --logdir /content/drive/My\ Drive/Keras3/classification_experiments/
```

In [12]:
```
import os
from datetime import datetime
cwd = os.getcwd()

exps_dir = os.path.join('/content/drive/My Drive/Keras3/', 'classification_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

model_name = 'CNN'

exp_dir = os.path.join(exps_dir, model_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# ----------------------------------------------------------------------------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ----------------------------------------------------------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=1)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# ----------------------------------------------------------------------------------
early_stop = True
if early_stop:
    es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
    callbacks.append(es_callback)
```

In [13]:
```
# Run this cell only if you want to plot the confusion matrix in tensorboard
import io
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

file_writer_cm = tf.summary.create_file_writer(tb_dir + '/cm')

# Function to convert input figure to png tf image (for plotting the confusion matrix in tensorboard)
def plot_to_image(figure):
  # Save the plot to a PNG in memory.
  buf = io.BytesIO()
  plt.savefig(buf, format='png')
  # Closing the figure prevents it from being displayed directly inside the notebook
  plt.close(figure)
  buf.seek(0)
  # Convert PNG buffer to TF image
  image = tf.image.decode_png(buf.getvalue(), channels=4)
  # Add the batch dimension
  image = tf.expand_dims(image, 0)
  return image

# Function to compute the confusion matrix (using sklearn) and to save it in tensorboard
def log_confusion_matrix(epoch, logs):
  Y_prediction = model.predict_generator(test_gen, len(test_gen))
  # Convert predictions classes to one hot vectors
  Y_pred_classes = np.argmax(Y_prediction,axis = 1)
  # Convert validation observations to one hot vectors
  Y_true = test_gen.classes
  # compute the confusion matrix
  confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
  fig = plt.figure(figsize=(10,8))
  sns.heatmap(confusion_mtx, annot=True, fmt="d");

  cm_image = plot_to_image(fig)

  with file_writer_cm.as_default():
    tf.summary.image("Confusion Matrix", cm_image, step=epoch)

cm_callback = tf.keras.callbacks.LambdaCallback(on_epoch_end=log_confusion_matrix)
callbacks.append(cm_callback)
```




In [14]:
```
model.fit(x=train_dataset,
```

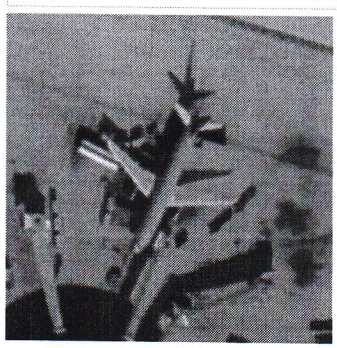
```
          epochs=10,  #### set repeat in training dataset
          steps_per_epoch=len(train_gen),
          validation_data=valid_dataset,
          validation_steps=len(valid_gen),
          callbacks=callbacks)

# How to visualize Tensorboard

# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/10
184/184 [==============================] - ETA: 0s - loss: 2.9386 - accuracy: 0.0810WARNING:tensorflow:From <ipython-inp
ut-13-3a6e0c298ce0>:25: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
184/184 [==============================] - 6s 32ms/step - loss: 2.9386 - accuracy: 0.0810 - val_loss: 2.6348 - val_accur
acy: 0.2032
Epoch 2/10
184/184 [==============================] - 6s 31ms/step - loss: 2.3648 - accuracy: 0.2694 - val_loss: 2.2446 - val_accur
acy: 0.2698
Epoch 3/10
184/184 [==============================] - 6s 32ms/step - loss: 2.0101 - accuracy: 0.3830 - val_loss: 1.8690 - val_accur
acy: 0.4000
Epoch 4/10
184/184 [==============================] - 6s 32ms/step - loss: 1.6496 - accuracy: 0.5048 - val_loss: 1.7998 - val_accur
acy: 0.4032
Epoch 5/10
184/184 [==============================] - 6s 32ms/step - loss: 1.3634 - accuracy: 0.5898 - val_loss: 1.5939 - val_accur
acy: 0.5111
Epoch 6/10
184/184 [==============================] - 6s 31ms/step - loss: 1.0970 - accuracy: 0.6619 - val_loss: 1.3677 - val_accur
acy: 0.5333
Epoch 7/10
184/184 [==============================] - 6s 31ms/step - loss: 0.8946 - accuracy: 0.7286 - val_loss: 1.2704 - val_accur
acy: 0.5746
Epoch 8/10
184/184 [==============================] - 6s 32ms/step - loss: 0.7143 - accuracy: 0.7796 - val_loss: 1.2403 - val_accur
acy: 0.5778
Epoch 9/10
184/184 [==============================] - 5s 29ms/step - loss: 0.5630 - accuracy: 0.8422 - val_loss: 1.3459 - val_accur
acy: 0.5619
Epoch 10/10
184/184 [==============================] - 6s 32ms/step - loss: 0.4719 - accuracy: 0.8571 - val_loss: 1.3449 - val_accur
acy: 0.6254
```

Out[14]: `<tensorflow.python.keras.callbacks.History at 0x7ff29f7a4940>`

In [17]:
```
# Let's visualize the activations of our network
from PIL import Image

test_iter = iter(test_dataset)

# Get a test image
test_img = next(test_iter)[0]
test_img = test_img[0]

# Visualize the image
Image.fromarray(np.uint8(np.array(test_img)*255.))
```

Out[17]:



In [18]:
```
# Get the activations (the output of each ReLU layer)
layer_outputs = [layer.output for layer in model.layers if isinstance(layer, tf.keras.layers.ReLU)]
# We can do it by creating a new model (activation_model) with model.input as input
# and all the ReLU activations as output
activation_model = tf.keras.Model(inputs=model.input, outputs=layer_outputs)
# Finally we get the output values given the imput test image
activations = activation_model.predict(tf.expand_dims(test_img, 0))
```

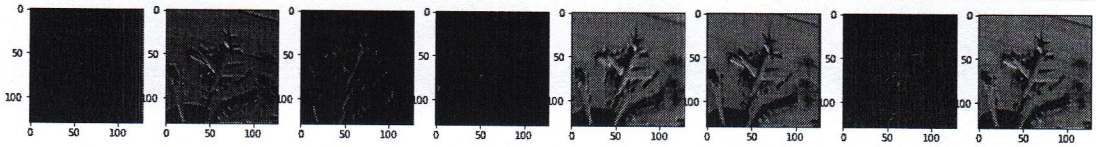
In [21]:
```
# Use this instead of the previous cell if you have implemented the custom classes

# Get the activations (the output of each ReLU layer in the feature_extractor)
activations = []
x = tf.expand_dims(test_img, 0)
for conv_block in model.feature_extractor.layers:
  for layer in conv_block.layers:
    x = layer(x)
    if isinstance(layer, tf.keras.layers.ReLU):
      activations.append(x)
```

(look for In[20])

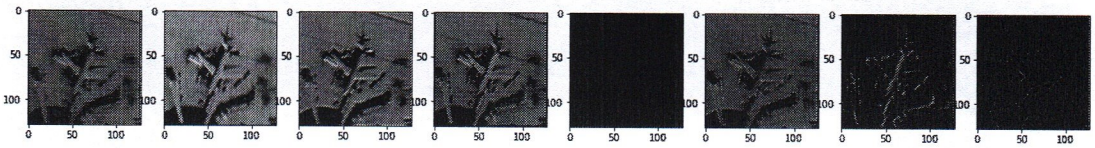
In [22]:
```
import matplotlib.pyplot as plt
%matplotlib inline
def display_activation(activations, act_index):
    # activations: list of all the activations
    # act_index: the layer we want to visualize (an int in [0, network depth) )
    activation = activations[act_index]
    activation = tf.image.resize(activation, size=[128, 128])
    col_size = activations[0].shape[-1]
    row_size = 1 + act_index
```

```
    activation_index=0
    fig, ax = plt.subplots(row_size, col_size, figsize=(8*2.5, 8*1.5), squeeze=False)
    for row in range(0,row_size):
        for col in range(0,col_size):
            ax[row][col].imshow(activation[0, :, :, activation_index], cmap='gray')
            activation_index += 1
```

In [23]:
```
display_activation(activations, 1)
```



The first activations are very recognizable. This means that they're capturing very superficial features.
The last activations will be non-interpretable.



In [24]:
```
# Print Confusion Matrix and Classification Report (Precision, Recall, and F1-score)
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

Y_prediction = model.predict_generator(test_gen, len(test_gen))
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_prediction,axis = 1)
# Convert validation observations to one hot vectors
Y_true = test_gen.classes
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
class_report = classification_report(Y_true, Y_pred_classes,
                                     target_names=test_gen.class_indices.keys())  # target_names must be ordered
                                                                                  # depending on the class labels
print('Confusion Matrix:')
print(confusion_mtx)
print()
print('Classification Report:')
print(class_report)
```

```
Confusion Matrix:
[[14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0]
 [10  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0  3  0  0]
 [11  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  2  0  0]
 [ 7  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  6  0  0]
 [ 7  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  7  0  0]
 [15  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0]
 [11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0]
 [12  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  2  0  0]
 [ 6  0  0  0  0  0  0  0  0  0  0  4  0  0  0  1  0  0  4  0  0]
 [ 9  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  5  0  0]
 [12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0]
 [ 6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  9  0  0]
 [13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0]
 [13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0]
 [ 9  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0]
 [ 8  0  0  1  0  0  0  0  0  0  0  4  0  0  0  0  0  0  2  0  0]
 [ 9  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  4  1  0]
 [13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0]
 [11  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  3  0  0]
 [11  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  2  0  0]]
```

← this is so bad because we trained the model too little!

```
Classification Report:
                    precision    recall  f1-score   support

       agricultural       0.06      0.93      0.12        15
           airplane       0.00      0.00      0.00        15
    baseballdiamond       0.00      0.00      0.00        15
              beach       0.00      0.00      0.00        15
          buildings       0.00      0.00      0.00        15
          chaparral       0.00      0.00      0.00        15
   denseresidential       0.00      0.00      0.00        15
             forest       0.00      0.00      0.00        15
            freeway       0.00      0.00      0.00        15
         golfcourse       0.00      0.00      0.00        15
             harbor       0.00      0.00      0.00        15
       intersection       0.00      0.00      0.00        15
  mediumresidential       0.00      0.00      0.00        15
     mobilehomepark       0.00      0.00      0.00        15
           overpass       0.00      0.00      0.00        15
        parkinglot       0.00      0.00      0.00        15
              river       0.00      0.00      0.00        15
             runway       0.00      0.00      0.00        15
  sparseresidential       0.03      0.13      0.05        15
       storagetanks       0.00      0.00      0.00        15
        tenniscourt       0.00      0.00      0.00        15

           accuracy                           0.05       315
          macro avg       0.00      0.05      0.01       315
       weighted avg       0.00      0.05      0.01       315
```

3/3

**Topics**: Transfer learning in CNN

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np
SEED = 1234
tf.random.set_seed(SEED)
cwd = os.getcwd()
```

# Example: Multi-class Land Use classification

## Uc Merced Land Use Dataset

In [3]:
```
# Run this cell only if you are using Colab with Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
# Run this cell only if you are using Colab with Drive
!unzip '/content/drive/My Drive/UCMerced_LandUse.zip'
```

In [5]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import preprocess_input

apply_data_augmentation = False

# Create training ImageDataGenerator object
if apply_data_augmentation:
    train_data_gen = ImageDataGenerator(rotation_range=10,
                                        width_shift_range=10,
                                        height_shift_range=10,
                                        zoom_range=0.3,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        fill_mode='constant',
                                        cval=0,
                                        preprocessing_function=preprocess_input)  # to apply vgg normalization
else:
    train_data_gen = ImageDataGenerator(preprocessing_function=preprocess_input)

# Create validation and test ImageDataGenerator objects
valid_data_gen = ImageDataGenerator(preprocessing_function=preprocess_input)
test_data_gen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

Some models are well pre-trained maybe using some specific normalization of the data. If we want to use the same normalization on our inputs (which is generally a good choice) we do: →

(doing this we use the function that has been used to normalize the inputs on our inputs)

In [6]:
```
# Create generators to read images from dataset directory
# -------------------------------------------------------------------------------
dataset_dir = os.path.join(cwd, 'UCMerced_LandUse')

bs = 8            # batch size
img_h = 256       # image shape (1/2)
img_w = 256       # image shape (2/2)
num_classes = 21  # number of classes

# Training
training_dir = os.path.join(dataset_dir, 'training')
train_gen = train_data_gen.flow_from_directory(training_dir,
                                               batch_size=bs,
                                               class_mode='categorical',
                                               shuffle=True,
                                               seed=SEED)  # targets are directly converted into one-hot vectors

# Validation
validation_dir = os.path.join(dataset_dir, 'validation')
valid_gen = valid_data_gen.flow_from_directory(validation_dir,
                                               batch_size=bs,
                                               class_mode='categorical',
                                               shuffle=False,
                                               seed=SEED)

# Test
test_dir = os.path.join(dataset_dir, 'test')
test_gen = test_data_gen.flow_from_directory(test_dir,
                                             batch_size=bs,
                                             class_mode='categorical',
                                             shuffle=False,
                                             seed=SEED)
```

```
Found 1470 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
Found 315 images belonging to 21 classes.
```

In [7]:
```
# Create Dataset objects
# -------------------------------------------------------------------------------
# Training
train_dataset = tf.data.Dataset.from_generator(lambda: train_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
train_dataset = train_dataset.repeat()

# Validation
valid_dataset = tf.data.Dataset.from_generator(lambda: valid_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
valid_dataset = valid_dataset.repeat()
```

```
# Test
test_dataset = tf.data.Dataset.from_generator(lambda: test_gen,
                                              output_types=(tf.float32, tf.float32),
                                              output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
test_dataset = valid_dataset.repeat()
```

pretrained on **IMAGE net**

The VGG is CNN + classifier. We discard the classifier and re-use the feature extractor

# Transfer Learning

## e.g., VGG16

when our dataset is similar to one used in an other already trained problem, we can avoid learning the features extractor from skretch. We can get advantage of an already pre-trained model to initialize the weights of the feature extractor.

we want only the feature extractor

In [8]:
```
# Load VGG16 Model
vgg = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=(img_h, img_w, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

In [9]:
```
vgg.summary()
```

```
Model: "vgg16"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 256, 256, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 256, 256, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 256, 256, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 128, 128, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 128, 128, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 128, 128, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 64, 64, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 64, 64, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 64, 64, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 64, 64, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 32, 32, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 32, 32, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 32, 32, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 32, 32, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 16, 16, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 8, 8, 512) | 0 |

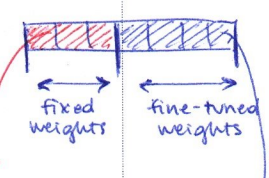
```
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

In a CNN : 

the first features extracted are very poor (low level features) and so we don't mind if we take the VGG16 's as they are. As we go deeper in the CNN (*) we want the weights to be very adapted to our problem, and so we want to fine-tune the weights to be right for the problem we're interested in (which can be very different from the original problem in VGG16).

In [11]:
```
# Create Model
# ----------------------------------------------------------------------

finetuning = True

if finetuning:
    freeze_until = 15 # Layer from which we want to fine-tune

    for layer in vgg.layers[:freeze_until]:
        layer.trainable = False
else:
    vgg.trainable = False

model = tf.keras.Sequential()
model.add(vgg)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=512, activation='relu'))
model.add(tf.keras.layers.Dense(units=num_classes, activation='softmax'))

# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

⇒ **CNN** of our problem :




here we do not update weights during optimization (= gradient descent)

here we do update weights

```
Model: "sequential_1"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| vgg16 (Functional) | (None, 8, 8, 512) | 14714688 |
| flatten_1 (Flatten) | (None, 32768) | 0 |
| dense_2 (Dense) | (None, 512) | 16777728 |
| dense_3 (Dense) | (None, 21) | 10773 |

```
Total params: 31,503,189
```

**Note**: we can also set the updating on false and we keep the exact same feature extractor of the VGG16

```
Trainable params: 23,867,925
Non-trainable params: 7,635,264
_________________________________________________________________
```

In [12]:
```
# Optimization params
# -------------------------------------------------------------------------------------
loss      = tf.keras.losses.CategoricalCrossentropy()   # Loss
lr        = 1e-4                                        # Learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)  # Optimizer

# Validation metrics
# -------------------------------------------------------------------------------------
metrics = ['accuracy']

# Compile Model
# -------------------------------------------------------------------------------------
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

In [14]:
```
import os
from datetime import datetime

# from tensorflow.compat.v1 import ConfigProto
# from tensorflow.compat.v1 import InteractiveSession

# config = ConfigProto()
# config.gpu_options.allow_growth = True
# session = InteractiveSession(config=config)

cwd = '/content/drive/My Drive/Keras3'

exps_dir = os.path.join(cwd, 'transfer_learning_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

model_name = 'CNN'

exp_dir = os.path.join(exps_dir, model_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# -------------------------------------------------------------------------------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# -------------------------------------------------------------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=1)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# -------------------------------------------------------------------------------------
early_stop = False
if early_stop:
    es_callback = tf.keras.callback.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)


model.fit(x=train_dataset,
          epochs=10,  #### set repeat in training dataset
          steps_per_epoch=len(train_gen),
          validation_data=valid_dataset,
          validation_steps=len(valid_gen),
          callbacks=callbacks)

# How to visualize Tensorboard

# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/10
184/184 [==============================] - 21s 112ms/step - loss: 1.7394 - accuracy: 0.5544 - val_loss: 0.8947 - val_acc
uracy: 0.7429
Epoch 2/10
184/184 [==============================] - 20s 111ms/step - loss: 0.4904 - accuracy: 0.8619 - val_loss: 0.8853 - val_acc
uracy: 0.7619
Epoch 3/10
184/184 [==============================] - 20s 108ms/step - loss: 0.4059 - accuracy: 0.9000 - val_loss: 0.8038 - val_acc
uracy: 0.7873
Epoch 4/10
184/184 [==============================] - 20s 109ms/step - loss: 0.1647 - accuracy: 0.9592 - val_loss: 0.5636 - val_acc
uracy: 0.8476
Epoch 5/10
184/184 [==============================] - 21s 112ms/step - loss: 0.0650 - accuracy: 0.9837 - val_loss: 0.6598 - val_acc
uracy: 0.8635
Epoch 6/10
184/184 [==============================] - 20s 109ms/step - loss: 0.0597 - accuracy: 0.9864 - val_loss: 0.6923 - val_acc
uracy: 0.8794
```

```
Epoch 7/10
184/184 [==============================] - 20s 110ms/step - loss: 0.2227 - accuracy: 0.9626 - val_loss: 1.8358 - val_acc
uracy: 0.7270
Epoch 8/10
184/184 [==============================] - 21s 112ms/step - loss: 0.7525 - accuracy: 0.8524 - val_loss: 1.2690 - val_acc
uracy: 0.7619
Epoch 9/10
184/184 [==============================] - 20s 109ms/step - loss: 0.1201 - accuracy: 0.9687 - val_loss: 1.0798 - val_acc
uracy: 0.8413
Epoch 10/10
184/184 [==============================] - 20s 111ms/step - loss: 0.2857 - accuracy: 0.9503 - val_loss: 1.3435 - val_acc
uracy: 0.8286
```

Out[14]: <tensorflow.python.keras.callbacks.History at 0x7fabf5155a20>

In [15]:
```
# Test model
# ------------------------------------------------------------------------------
# model.load_weights('/path/to/checkpoint')  # use this if you want to restore saved model

eval_out = model.evaluate(x=test_dataset,
                          steps=len(test_gen),
                          verbose=0)
eval_out
```

Out[15]: [1.3434981107711792, 0.828571438789367]