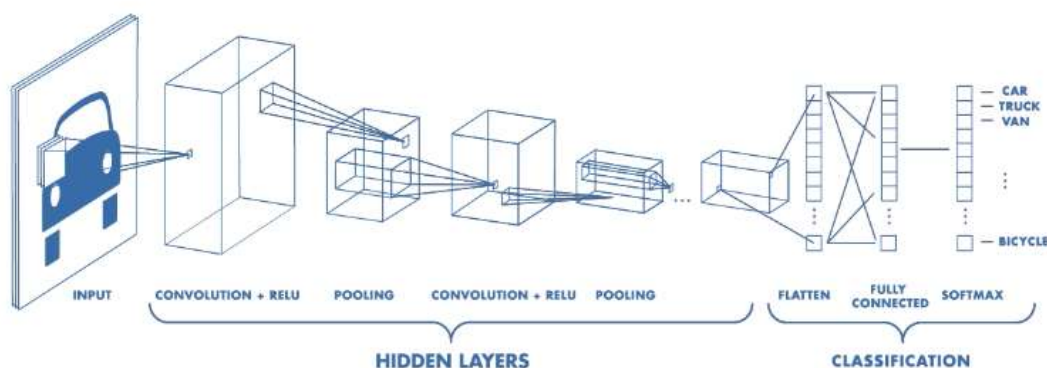


International Workshop on Applied Deep Learning

16th July – 20th July 2018



Day2 Hands-on Workbook

Convolutional Neural Networks

Learning Outcomes:

- How to load data from images
- How to define CNN model in Keras
- How to play along with Kernel initialisers and regularizers as well as parameters of optimisers
- How to add dropout in the model
- How to save and load models from checkpoints

Organiser: Dr. Aditya Nigam
www.faculty.iitmandi.ac.in/~aditya

1 Convolutional Neural Network - Part A

1.1 Load Libraries

```
import numpy as np

from keras.models import Sequential

from keras.layers import Dense , Conv2D, Dropout, Flatten, MaxPooling2D

from keras.utils import np_utils

from keras import regularizers

from keras.optimizers import SGD, Adam

from keras.callbacks import ModelCheckpoint

import h5py

import os

import cv2

from sklearn.model_selection import train_test_split

seed = 7

np.random.seed(seed)

#
```

1.2 Load Dataset from Images

In the root directory, there is a 'Data' folder containing separate folders 'train' and 'test'. 'train' and 'test' contains training data and testing data respectively. Inside these, we have multiple folders each corresponding to a particular class. And each of them contains images of that class. In the given dataset, we have 10 folders each in 'train' and 'test' corresponding to the 10 classes (0,1,...,9). So our task is to extract the data from images into numpy arrays.

```
def data_loader(path_train,path_test):
    train_list=os.listdir(path_train)

    num_classes=len(train_list)

    x_train=[]
```

```

y_train=[]

x_test=[]

y_test=[]

for label,elem in enumerate(train_list):

    path1=path_train+'/'+str(elem)

    images=os.listdir(path1)

    for elem2 in images:

        path2=path1+'/'+str(elem2)

        img = cv2.imread(path2)

        x_train.append(img)

        y_train.append(str(label))

    path1=path_test+'/'+str(elem)

    images=os.listdir(path1)

    for elem2 in images:

        path2=path1+'/'+str(elem2)

        img = cv2.imread(path2)

        x_test.append(img)

        y_test.append(str(label))

x_train=np.asarray(x_train)

y_train=np.asarray(y_train)

x_test=np.asarray(x_test)

y_test=np.asarray(y_test)

```

```
return x_train,y_train,x_test,y_test
```

```
#
```

1.3 Formatting Data and Labels for Deep Learning

```
input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
```

```
X_train = X_train.astype('float32')
```

```
X_test = X_test.astype('float32')
```

```
X_train = X_train / 255.
```

```
X_test = X_test / 255.
```

```
y_train = np_utils.to_categorical(y_train)
```

```
y_test = np_utils.to_categorical(y_test)
```

```
num_classes = y_test.shape[1]
```

```
#
```

1.4 Splitting Data into Training, Testing and Validation

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,  
↪ random_state=42)
```

```
#
```

1.5 Defining a Shallow CNN Model

Here we define a small CNN network. It contains a convolutional layer having 32 filters each of size (3×3) . Then we flatten the feature map and add 2 dense layers. First one having 100 neurons and last one having 10 neurons i.e. equal to number of classes.

```
model = Sequential()
```

```
model.add(Conv2D(32, (3,3), strides = (1,1), padding = 'same' , input_shape=input_shape,  
↪ activation='relu'))
```

```
model.add(Flatten())
```

```
model.add(Dense(100, activation='relu'))

model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))

#
```

1.6 Compiling the Model

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#
```

1.7 Training/Fitting the Model

```
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=2, batch_size=200,
↪ verbose=0)

#
```

1.8 Evaluating the Model

```
scores = model.evaluate(X_test, y_test, verbose=0)

print("Baseline Error: %.2f%%" % (100-scores[1]*100))

#
```

2 Convolutional Neural Network - Part B

Now we define a deeper model. It contains 2 blocks. Each block has 2 convolutional layers of 3×3 filter. The first block has 32 such filters in both of its layers. The second has 64 filters. Between them there is a max pooling layer. After the blocks, the feature map is flattened and few FC layers are added for classification task.

2.1 Defining a Deep Model

```
model = Sequential()

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same' , input_shape =
↪ input_shape, activation = 'relu'))

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))
```

```

model.add(MaxPooling2D((2, 2), strides=(2, 2), padding = 'valid'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))

model.add(Flatten())

model.add(Dense(500, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

#

```

2.2 Analyzing Model Summary

The following function provides us with detailed summary of the model. We can use it after we have defined our model.

```

model.summary()

```

```

#

```

2.3 Defining Kernel Initializes

```

model = Sequential()

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same' , input_shape =
↪ input_shape, activation = 'relu', kernel_initializer = 'glorot_uniform'))

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_initializer = 'glorot_uniform'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), padding = 'valid'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_initializer = 'glorot_uniform'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_initializer = 'glorot_uniform'))

model.add(Flatten())

```

```

model.add(Dense(500, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))

#

```

2.4 Defining Kernel Regularizer

```

model = Sequential()

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same' , input_shape =
↪ input_shape, activation = 'relu', kernel_regularizer = regularizers.l2(0.01)))

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_regularizer = regularizers.l2(0.01)))

model.add(MaxPooling2D((2, 2), strides=(2, 2), padding = 'valid'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_regularizer = regularizers.l2(0.01)))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu',
↪ kernel_regularizer = regularizers.l2(0.01)))

model.add(Flatten())

model.add(Dense(500, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

#

```

2.5 Adding Dropout to the Model

```

model = Sequential()

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same' , input_shape =
↪ input_shape, activation = 'relu'))

model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))

```

```

model.add(MaxPooling2D((2, 2), strides=(2, 2), padding = 'valid'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))

model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation = 'relu'))

model.add(Flatten())

model.add(Dense(500, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(100, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))

#

```

2.6 Defining Learning Rate Decay and Other Parameters of Optimizer

```

sgd = SGD(lr = 0.001, momentum = 0.0005, decay = 0.0005)

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0005)

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

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

#

```

2.7 Using Checkpoints and Saving/Loading the Model

```

model.save_weights('./CNN.h5')

model.load_weights('./CNN.h5')

filepath='./CNN.h5'

checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True,
↪ mode='max')

```



```
callbacks_list = [checkpoint]
```

```
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=2, batch_size=200,  
↪ callbacks=callbacks_list, verbose=2)
```

```
#
```

2.8 Programming Task

```
import numpy as np  
from keras.models import Sequential  
from keras.layers import Dense , Conv2D, Dropout, Flatten, MaxPooling2D  
from keras.utils import np_utils  
from keras import regularizers  
from keras.optimizers import SGD, Adam  
from keras.callbacks import ModelCheckpoint  
import h5py  
import os  
import cv2  
from sklearn.model_selection import train_test_split  
# fix random seed for reproducibility  
#we always initialize the random number generator to a constant seed #value for  
↪ reproducibility of results.  
seed = 7  
np.random.seed(seed)  
  
# load data from the path specified by the user  
def data_loader(path_train,path_test):  
    train_list=os.listdir(path_train)  
    # Number of classes in the dataset  
    num_classes=len(train_list)  
  
    # Empty lists for loading training and testing data images as well as corresponding  
    ↪ labels  
    x_train=[]  
    y_train=[]  
    x_test=[]  
    y_test=[]  
  
    # Loading training data  
    for label,elem in enumerate(train_list):  
  
        path1=path_train+'/'+str(elem)  
        images=os.listdir(path1)
```

```

    for elem2 in images:
        path2=path1+'/'+str(elem2)
        # Read the image form the directory
        img = cv2.imread(path2)
        # Append image to the train data list
        x_train.append(img)
        # Append class-label corresponding to the image
        y_train.append(str(label))

# Loading testing data
path1=path_test+'/'+str(elem)
images=os.listdir(path1)
for elem2 in images:
    path2=path1+'/'+str(elem2)
    # Read the image form the directory
    img = cv2.imread(path2)
    # Append image to the test data list
    x_test.append(img)
    # Append class-label corresponding to the image
    y_test.append(str(label))

# Convert lists into numpy arrays
x_train=np.asarray(x_train)
y_train=np.asarray(y_train)
x_test=np.asarray(x_test)
y_test=np.asarray(y_test)
return x_train,y_train,x_test,y_test

path_train='./Data/train'
path_test='./Data/test'

X_train,y_train,X_test,y_test=data_loader(path_train,path_test)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
# forcing the precision of the pixel values to be 32 bit
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255.
X_test = X_test / 255.
# one hot encode outputs using np_utils.to_categorical inbuilt function
y_train = np_utils.to_categorical(y_train)

```

```

y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

#Splitting the training data into training and validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
↪ random_state=42)

# define baseline model
#The model is a simple neural network with one hidden layer with the same number of
↪ neurons as there are inputs (784)
def baseline_model():
    # create model
    model = Sequential()
    #We will add 2 Convolution layers with 32 filters of 3x3, keeping the padding as
    ↪ same
    model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same' , input_shape =
    ↪ input_shape, activation = 'relu', kernel_initializer = 'glorot_uniform',
    ↪ kernel_regularizer = regularizers.l2(0.01)))
    model.add(Conv2D(32, (3, 3), strides=(1, 1), padding = 'same', activation =
    ↪ 'relu', kernel_initializer = 'glorot_uniform', kernel_regularizer =
    ↪ regularizers.l2(0.01)))
    #Pooling the feature map using a 2x2 pool filter
    model.add(MaxPooling2D((2, 2), strides=(2, 2), padding = 'valid'))
    #Adding 2 more Convolutional layers having 64 filters of 3x3
    model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation =
    ↪ 'relu', kernel_initializer = 'glorot_uniform', kernel_regularizer =
    ↪ regularizers.l2(0.01)))
    model.add(Conv2D(64, (3, 3), strides=(1, 1), padding = 'same', activation =
    ↪ 'relu', kernel_initializer = 'glorot_uniform', kernel_regularizer =
    ↪ regularizers.l2(0.01)))
    #Flatten the feature map
    model.add(Flatten())
    #Adding FC Layers
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(100, activation='relu'))
    model.add(Dropout(0.3))
    #A softmax activation function is used on the output
    #to turn the outputs into probability-like values and
    #allow one class of the 10 to be selected as the model's output #prediction.
    model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
    #Checking the model summary
    model.summary()
    # Loading weights
    model.load_weights('./CNN.h5')

```

```

# Compile model
sgd = SGD(lr = 0.001, momentum = 0.0005, decay = 0.0005)
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=adam,
    ↪ metrics=['accuracy'])
#model.compile(loss='categorical_crossentropy', optimizer=sgd,
    ↪ metrics=['accuracy'])
return model

# build the model
model = baseline_model()

# Fit the model
#The model is fit over 10 epochs with updates every 200 images. The test data is used as
    ↪ the validation dataset
# checkpoint
filepath='./CNN.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True,
    ↪ mode='max')
callbacks_list = [checkpoint]
# Fit the model
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=2, batch_size=200,
    ↪ callbacks=callbacks_list, verbose=2)
#Saving the model
model.save_weights('./CNN.h5')
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))

```
