Department of Computer Engineering
Faculty of Engineering, University of Peradeniya
CO542
Neural Networks and Fuzzy Systems
2021
Lab 06 – Convolutional Neural Networks (CNN)

Objectives

- Implement a Convolutional Neural Network using modern python based frameworks (Kears in this case).
- Study and practice concepts associated with CNNs (e.g.: convolution, pooling, dropout etc.).
- Identify the real-world usages of CNN.

Convolutional Neural Networks

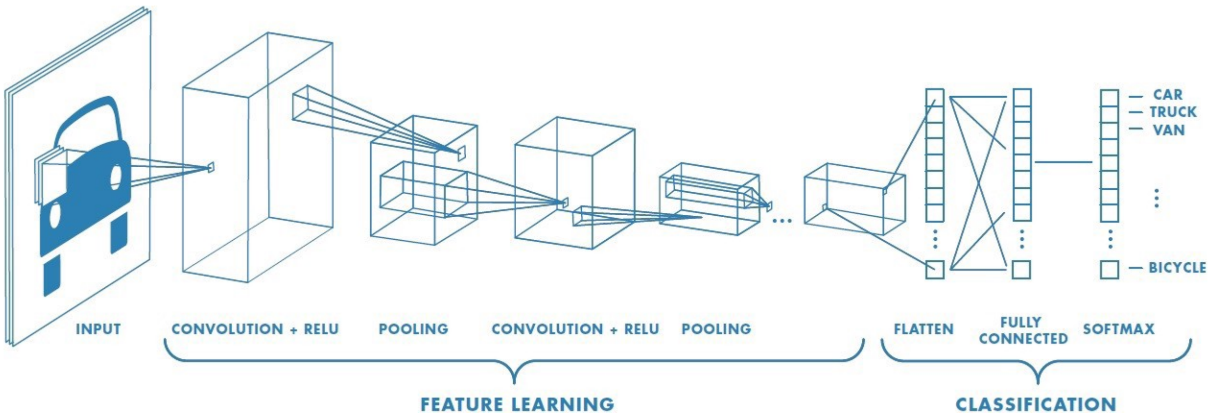


Figure 1: Convolutional Neural Networks (Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

Guide for Implementing CNN Using Keras

In this case, we are going to use one of the popular datasets known as 'mnist' which contains 70000 images of handwritten digits from 1-9. Therefore, out task is to identify them using a CNN.

1. ***Step 1:*** *Loading the dataset*
   The mnist dataset is conveniently provided to us as part of the Keras library, so we can easily load the dataset. Out of the 70,000 images provided in the dataset, 60,000 are given for training and 10,000 are given for testing.

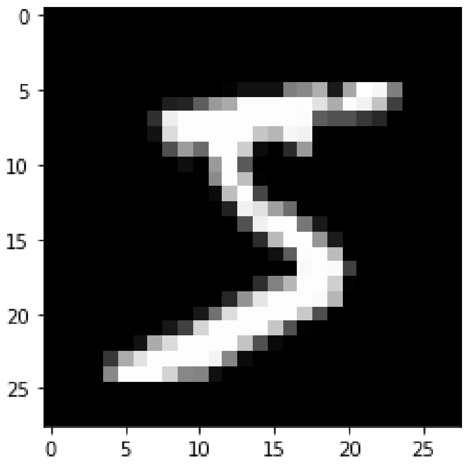
   ```
   from keras.datasets import mnist

   #download mnist data and split into train and test sets
   (X_train, y_train), (X_test, y_test) = mnist.load_data()
   ```

2. ***Step 2:*** *Exploratory data analysis*
   Now let's take a look at one of the images in our dataset to see what we are working with. We will plot the first image in our dataset and check its size using the 'shape' function.

   ```
   import matplotlib.pyplot as plt

   #plot the first image in the dataset
   plt.imshow(X_train[0], cmap='gray')
   ```



   ```
   #check image shape
   X_train[0].shape
   ```

```
(28, 28)
```

3. ***Step 3:*** *Data pre-processing*
   Next, we need to reshape our dataset inputs (X_train and X_test) to the shape that our model expects when we train the model. The first number is the number of images (60,000 for X_train and 10,000 for X_test). Then comes the shape of each image (28x28). The last number is 1, which signifies that the images are grey-scale.

   ```
   #reshape data to fit model
   X_train = X_train.reshape(60000,28,28,1)
   X_test = X_test.reshape(10000,28,28,1)
   ```

   We need to 'one-hot-encode' our target variable.

   ```
   from keras.utils import to_categorical

   #one-hot encode target column
   y_train = to_categorical(y_train)
   y_test = to_categorical(y_test)
   y_train[0]

       array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
   ```

   Split the dataset for training and validation purposes

   ```
   from sklearn.model_selection import train_test_split
   train_X, valid_X, train_label, valid_label = train_test_split(X_train,
                                                                 y_train,
                                                                 test_size=0.2,
                                                                 random_state=13)
   ```

4. ***Step 4:*** *Building the model*
   Now we are ready to build our model. Here is the code:

   ```
   from keras.models import Sequential
   from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout

   #create model
   model = Sequential()

   #add model layers
   model.add(Conv2D(64, kernel_size=3, activation='relu',
                                    input_shape=(28,28,1)))
   model.add(MaxPooling2D((2,2), padding='same'))
   model.add(Dropout(0.25))

   model.add(Conv2D(32, kernel_size=3, activation='relu'))
   model.add(MaxPooling2D((2,2), padding='same'))
   model.add(Dropout(0.25))

   model.add(Flatten())

   model.add(Dense(10, activation='softmax'))
   ```

5. ***Step 5:*** *Compiling the model*
   Next, we need to compile our model. Compiling the model takes three parameters: optimizer, loss and metrics.

```
#compile model using accuracy to measure model performance
model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

6. ***Step 6:*** *Training the model*
   Now we will train our model. To train, we will use the 'fit()' function on our model with the following parameters: training data (train_X), target data (train_label), validation data, and the number of epochs.

   ```
   #train the model
   model.fit(train_X, train_label, validation_data=(valid_X, valid_label),
                                                   epochs=3)
   ```

7. ***Step 7:*** *Test the trained model*
   Then we have to evaluate the model to measure its performance. Use the following code.

   ```
   test_eval = model.evaluate(X_test, y_test, verbose=0)
   print("Test loss:", test_eval[0])
   print("Test accuracy:", test_eval[1])
   ```

8. ***Step 8:*** *Using our model to make predictions*
   If you want to see the actual predictions that our model has made for the test data, we can use the predict function.

   ```
   predictions = model.predict(X_test)
   import numpy as np
   predictions = np.argmax(np.round(predictions),axis=1)
   predictions[:5]

                           array([7, 2, 1, 0, 4])
   ```

Exercise

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck) with 6000 images per class. There are 50000 training images and 10000 test images. Your task here is to build and train a CNN to detect images in each of these classes using Keras framework and other required libraries accordingly.

1. Import the CIFER-10 data set using keras.datasets.
2. Study the shapes of the training and testing datasets.
3. Visualize some images in the train and test tests to understand the dataset. You may use matplotlib.pyplot.imshow to display the images in a grid.
4. Under the data pre-processing procedures,
   - Reshape the input datasets accordingly.
   - Normalize the pixel values in a range between 0 to 1.
   - Convert the class labels into One-Hot encoding vector. Clearly mention the requirement of this conversion.

- Use sklearn.model_selection.train_test_split to further split the training dataset into validation and training data (e.g. allocate 0.2 of the training set as validation data).

5. Build the CNN model with three convolutional layers followed by a dense layer and an output layer accordingly. In this case,
   - Select 3 X 3 as the kernal size of each filter.
   - Use different number of filters in each convolutional layer (e.g. first layer 32 filters, second layer 64 filters, third layer 128 filters).
   - Use LeakyReLU as the activation function. Mention the advantage of using LeakyReLU over ReLU activation function.
   - Use 2 X 2 MaxPooling layers, and Dropout layers according to the requirements and mention the purpose behind the usage of Dropout Layers.
6. Compile the model using appropriate parameters and generate the model summery using model.summary() function (In this case make sure to specify the metrics as accuracy).
7. Train the compiled model using model.fit function and observe the train and validation set performances. In this case, you may have to select an appropriate number of epochs (e.g. 25) and batch_size (e.g. 64, 128 or 256).
8. Evaluate the model performance using test set. Identify the test loss and test accuracy.
9. Use the trained model to make predictions for the test data and visualize the model performance under each class using sklearn.metrics.classification_report.

Submission

Make sure to submit all the code files (either .py or .ipynb format) and a PDF file containing,

- All the expansions
- Generated summaries /reports and tables.
- Visualized images
- Predictions
- All the other necessary outputs etc...

in a single .zip file before the deadline mentioned in the course page.

Note: Please submit all the files (pdfs, code files etc.) in a single zipped file renamed as EXXYYY_Lab06, where XX is your batch ID, and YYY is your E No.

Reference

- https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
- https://keras.io/about/
- https://www.cs.toronto.edu/ kriz/cifar.html
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks -the-eli5-way-3bd2b1164a53
- https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python