Generated at Sat Dec 10 11:38:16 2022.

Source at https://jupyterhub-dev.cheme.cmu.edu/user/sjilhewa@andrew.cmu.edu/lab/tree/f22-06623/assignments/project-checkin/project-checkin-Copy1.ipynb.

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:

```
NAME = "Saaksshi Jilhewar"
COLLABORATORS = ""
```

Convolutional Neural Networks

Artificial Intelligence has been revolutionizing the gap between the adoption of human capabilities by machines. One sector that is really challenging and is an active area of research is “Computer Vision”.

The challenge for this field is to enable machines to view the world as humans do, perceive it in a similar manner, and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning have been constructed and perfected with time, primarily over one particular algorithm — a Convolutional Neural Network. [1]

image.png

In a CNN we feed images flowers in the above case to the input layer and then we add Conv layer with ReLU activation followed by pool and this all consists of hidden layers, later we add an output layer which gives us the prediction, type of flower in this case.

Introduction to Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that takes an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and is able to perform identification(differentiation task). The pre-processing required in a ConvNet is much lower as compared to other classification algorithms, which is the main reason to add ConvNet in Artificial Neural Network Algorithm because when we have to perform these tasks on images with better resolution the computation complexity keeps increasing with the increase in the number of pixels. The use of ConvNets helps to do feature recognition in any image and to learn these characteristics, with less computational complexity.[2]

Thinking from the computer’s perspective, an image can be described as a collection of 3 different matrices with the Red, Blue, and Green characteristics of the image into one matrix. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. The role of ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction. This is important when we are to design an architecture that is not only good at learning features but also scalable to massive datasets.

Why can’t we use Artificial Neural Networks for the same purpose?

This is because there are some disadvantages with ANN:

It is too much computation for an ANN model to train large-size images and different types of image channels.

The next disadvantage is that it is unable to capture all the information from an image whereas a CNN model can capture the spatial dependencies of the image.

Another reason is that ANN is sensitive to the location of the object in the image i.e if the location or place of the same object changes, it will not be able to classify properly. [3]

Components of CNN

Scanner / Filter / Kernel

CNN uses filters, and each filter extracts some information from the image such as edges, and different kinds of shapes (vertical, horizontal, round), and then all of these are combined to identify the image. The kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products. This process is called convolution.

The image shows the implimenation, Convolution with 3×3 Filter, Source: http://deeplearning.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/ image.png

The kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products. Kernel moves on the input data by the stride value. The kernel extracts the high-level features like edges from the image. Learning of the weights in the kernel is learned by back propogation. CNN are automatic feature detectors, the network itself while training learns the filter weights and bias terms.

How kernel works on a 3-D image: image.png In a 3D image we have 3 channels red, green and blue. To scan over these channels we use a 3D cube kernel more like a 3D matrix that sweeps over the image cube and scans the image to give us an output performing convolution.

Architecture

image.png

The architecture consists of channels that are derived after the kernel scans the images, then a pool layer and then again a Conv layer where the again the features are extracted followed by pool layer. Finally Flattening the whole channels into a Fully Connected layer which then forms neural network, followed by an output layer. The neurons in the output layer depends on the type of task we CNN is performing Classification/Regression.

Components of CNN:

Input Image -> Convolution(Feature detection) -> Non-Linearity -> Spatial Pooling -> Feature maps -> Fully connected layers -> Output layer

Activation Function

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. ReLU function is the most widely used activation function in neural networks today. One of the greatest advantage ReLU has over other activation functions is that it does not activate all neurons at the same time. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions. Some disadvantage ReLU presents is that it is saturated at the negative region, meaning that the gradient at that region is zero. With the gradient equal to zero, during backpropagation all the weights will not be updated, to fix this, we use Leaky ReLU. Also, ReLU functions are not zero-centered. This means that for it to get to its optimal point, it will have to use a zig-zag path which may be longer.[4] We use ReLU in hidden layer to avoid vanishing gradient problem and better computation performance , and Softmax function use in last output layer .

image.png

Image source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

image.png

Image source: https://www.nomidl.com/deep-learning/what-is-the-difference-between-sigmoid-and-softmax-activation-function/

Pooling

Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust. Sample types of pooling are max pooling, avg pooling, and sum pooling, but these days max pooling is more common. The Pooling layer can be seen between Convolution layers in a CNN architecture. This layer basically reduces the number of parameters and computation in the network, controlling over fitting by progressively reducing the spatial size of the network.There are two operations in this layer: Average pooling and Maximum pooling. Max-pooling, is like the name states; we will take out only the maximum from a pool. This is actually done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped. This actually down-samples the network. Unlike the convolution layer, the pooling layer does not alter the depth of the network, the depth dimension remains unchanged. [5]

image.png

Padding

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN. adding is a process of adding zeros to the input matrix symmetrically. It is used to make the dimension of output same as input.

When the kernel scans, it does affect the edge parameters since they don't get scanned as much as the inner parameters gets. This may lead to loss of information on the edges which can be an important case. For situations like these we add padding to ensuring that the edge parameters are now inner parameters. [5]

Padding has the following benefits:

1. It allows us to use a Conv layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels on the edges of an image.
3. We have a dimensional constraint on CNN: the dimension of an output layer is (N - F)/S + 1, where N is the dimension of the input, F is the dimension of the filter, and S is the stride. Under this constraint, certain inputs cannot be performed CNN. For instance, if we have N = 7, F = 3, S = 3, the output dimension is non-integer. Therefore, we need a tool to fix dimension disagreement.

Practical Implementation of CNN

Example 1 : Building a simple CNN - Classification Task

Let's see how to build a simple neural network. To build CNN here, we have used Keras, which is a neural network library that runs on top of the TensorFLow, which is an open-sourced platform to perform machine learning tasks.

MNIST:

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.

In [2]:

```
#importing the required libraries

from tensorflow.keras.datasets import mnist #importing dataset which is already available in keras
from tensorflow.keras.models import Sequential #used to build the model
from tensorflow.keras.layers import Conv2D #defining 2D Conv layer
from tensorflow.keras.layers import MaxPool2D #max pool layer
from tensorflow.keras.layers import Flatten #flattening layer
from tensorflow.keras.layers import Dense #reduction layers

from matplotlib import pyplot
```

Here,we have used the mnist dataset which is already available in keras library.

While loading the dataset we load it as train and test differently, so we can use the train to train our model and test to test the trained model with.

X_train contains the training dataset and y_train contains the output which we learn from all the images basically prediction, we train the model using X_train and y_train and later we feed X_test to the model compare the y_test which is the original output also called as ground truth, compare the model's prediction with our ground truth and check the accuracy.

In [3]:

```
#loading data and splitting the dataset
(X_train,y_train), (X_test,y_test) = mnist.load_data()

#reshaping data
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
X_test = X_test.reshape((X_test.shape[0],X_test.shape[1],X_test.shape[2],1))
```

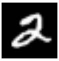
```
#checking the shape after reshaping
print("shape of mnist dataset")
print(X_train.shape)
print(X_test.shape)
print("\n")
```

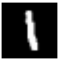
```
shape of mnist dataset
(60000, 28, 28, 1)
(10000, 28, 28, 1)
```

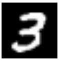
In [4]:

```
#printing the mnist dataset
print("mnist dataset")
for i in range(3):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(X_train[i+5], cmap=pyplot.get_cmap('gray'))
    pyplot.axis('off')
    pyplot.show()
```

```
mnist dataset
```







We need to understand what we are doing before proceeding further. We have a dataset which is a collection of hand written numbers 0-9, using which we will train our model to predict the number what is in the image. This is a multiclass classification problem, since we have 10 different prediction which are : 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

To build the model we need to use Sequential and Dense to execute Deep Learning which is available under the Keras library, which we have already imported.

We store the Sequential function to model, so as of now this model is our network where we can add and and perform all the tasks that's needed.

The output layer has exactly 10 neurons, which is a reflection of the classes we have.

Here, we have used 'relu' as my activation function and 'softmax' as an activation function for my last layer. We use 'cross-entropy' our loss function since it's a multi class classification problem, to optimize our model we use 'adam' as our optimizer and 'accuracy' as our metrics to evaluate our model.

In [5]:

```
#defining model
model = Sequential()

#adding convolution layer
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))

#adding pooling layer
model.add(MaxPool2D(2,2))

#adding fully connected layer
model.add(Flatten())
model.add(Dense(1000,activation='relu'))
model.add(Dense(100,activation='relu'))

#adding output layer
model.add(Dense(10,activation='softmax'))

#compiling the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

To get an overview of our model we use 'model.summary()', which provides brief details about our model.

In [6]:

```
#model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0         
 )                                                               
                                                                 
 flatten (Flatten)           (None, 5408)              0         
                                                                 
 dense (Dense)               (None, 1000)              5409000   
                                                                 
 dense_1 (Dense)             (None, 100)               100100    
                                                                 
 dense_2 (Dense)             (None, 10)                1010      
                                                                 
=================================================================
Total params: 5,510,430
Trainable params: 5,510,430
Non-trainable params: 0
_________________________________________________________________
```

Understanding the summary output:

Our first layer is a convolution, which takes an unknown input shape (it's known by us, we defined it with input_shape=....

The first convolution has an output with shape (None, 26, 26, 32), where:

None is the batch size.

26 and 26 are the size of the resulting image.

32 are the number of filters of this convolution and also the number of channels in its output.

Then we have a added maxpooling layer that takes the output of the convolution as input. The output of the pooling has shape (None, 13, 13, 32), so it divided the size of our image by two, leaving the rest as it was.

Then we have a Flatten layer, which takes the images and transform them into a single vector, output shape is (None, 5408), where None is still the batch size untouched, the 5408 are all elements we had in the input tensor as single vector.

Then Dense layers, the first with 1000 units, the second with 100 units and the third with 10 units.

The final output shape of your model is (None, 10). It outputs 10 values per sample in the batch.

Each layer has a number of parameters (which are generally the weights). The parameters that are trainable will be updated with back propagation. The parameters that are not trainable will remain static or will be updated with a different method (only a few layers such as Batch Normalization has parameters that are updated with different methods)

Your model has a total of 5,510,430 weights, all trainable.

Now, we have the model built, we'll fit our training dataset using '.fit()'.

We have added epoch here, epoch define the training set of data. When we set epoch = 10 we mean that we are training the dataset 10 times. We do so to get better accuracy, because the network can perform differently and learn different weights and biases, the more we train our model becomes good at distinguishing and learns the parameter better, but we need to consider the concept of over fitting as well.

In [ ]:

```
#fitting the model
model.fit(X_train,y_train, epochs = 10)
```

```
Epoch 1/10
1875/1875 [==============================] - 29s 15ms/step - loss: 0.7277 - accuracy: 0.9237
Epoch 2/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.0736 - accuracy: 0.9787
Epoch 3/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0516 - accuracy: 0.9846
Epoch 4/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0390 - accuracy: 0.9881
Epoch 5/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0337 - accuracy: 0.9901
Epoch 6/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0278 - accuracy: 0.9918
Epoch 7/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0218 - accuracy: 0.9941
Epoch 8/10
1130/1875 [=================>............] - ETA: 10s - loss: 0.0206 - accuracy: 0.9946
```

We can see initially in it's leaning phase the model, had an accuracy of 96.23% but with increase in epoch it also increased, with significant decrease in loss.

Now, that we have trained our model we use '.evaluate()' to predict, we need to understand while training the model we give the whole dataset to the model and it learns the parameters to predict the given ground truth, but when we evaluate model we just give the inputs and the output is the prediction by the model, which is done by using the learned parameters while the training phase. So, when we evaluate the model we only feed X_test to the, get it's prediction compare to the ground truth and then evaluate it. In this case on the basis of accuracy.

In [ ]:

```
#evaluting the model
accuracy = model.evaluate(X_test,y_test)
```

We have stored the accuracy in the 'accuracy' variable, but if we check the shape of the variable we can see it has a shape of 2, this is because it saves loss and accuracy.

In [ ]:

```
import numpy as np
np.shape(accuracy)
```

In [ ]:

```
print(f"Accuracy: {accuracy[1]*100:1.3}%")
```

Finally, we have achieved the result and we secured an accuracy of 98.7% in the test set which is very much appreciable, this can also be considered as the performance of our model. We can say that given any new dataset to predict the number the model can do it with 98.7% accuracy.

Example 2 : Prediction of Glass Temperature - Regression Task

So far we have learned to built a CNN architecture, along with understanding the layers. Now, in this example we will learn to create images, this example is inspired from From chemical structure to quantitative polymer properties prediction through convolutional neural networks. [6] The dataset for this problem was also referred from the same.
image.png

Here, in this example we will build our own images from the dataset and then build a CNN architecture for prediction of Glass Temperature of polymers.

Importing libraries and datset

In [ ]:

```
# !pip install tensorflow
# !pip install tensorflow --ignore-installed --user
```

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,MaxPool2D,Conv2D,Conv1D,Flatten,MaxPooling1D
from keras import optimizers
from keras.layers import Dropout
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
```

Generating image from dataset

The dataset consists of different SMILES (simplified molecular-input line-entry system) which uses short ASCII string to represent the structure of chemical species codes for different polymers along with it's Glass temperature.

The datset contains some nan values, so we use '.dropna' to drop all rows containing those null/nan values.

In [ ]:

```
#ingesting data
df=pd.read_csv("molecules.csv", encoding='windows-1254')

#droping nan
df=df.dropna(how="all")

#declaring the dataframe to be used
df=df.iloc[:,:3]
df["Nomenclature Name"]=df.iloc[:,2]
df["Molecular Structure"]=df.iloc[:,1]
df["Tg"]=df.iloc[:,0]
df=df[["Nomenclature Name","Molecular Structure","Tg"]]
df.head(10)
```

We define a function which encodes all the SMILES code into one hot encoding according to the elements present in the SMILES code. We will make matrix of all the encoded against the number of elements we used. This function here will return matrices.

In [ ]:

```
def encoding(df):
    encoded=[]
    elements=[['c'], ['n'], ['o'], ['C'], ['N'], ['F'], ['='], ['O'],
            ['('], [')'], ['1'],['2'],['#'],['Cl'],['/']]
    enc = OneHotEncoder(handle_unknown='ignore')
    enc.fit(elements)
    enc.categories_
    df1=df["Molecular Structure"].apply(lambda x: pd.Series(list(x)))
    for i in range(df1.shape[0]):
        x=enc.transform(pd.DataFrame(df1.iloc[i,:]).dropna(how="all").values).toarray()
        y=np.zeros(((df1.shape[1]-x.shape[0]),len(elements)))
        encoded.append(np.vstack((x,y)))
    return encoded
```

Once we have the matrix, we can convert this matrix into 2-D images, according to the number in the matrix, it will get it's pixel intensity of that pixel.

In [ ]:

```
def encoded_generate_images(df):
    listt=encoding(df)
    plt.figure(figsize=(20,100))
    for i in range(len(listt)):
        plt.subplot(len(listt),5,i+1)
        plt.imshow(listt[i])
        plt.axis('off')
```

Let us see how the images look like.

In [ ]:

```
encoded_generate_images(df.head(10))
```

The generated one hot encoded image takes into account the chemical structure and the composition of the monomeric unit. We can see that the encoded image tells us the number of each kind of atoms present in the monomeric structure in a binary form along with the alignment structure of atoms in the polymeric chain relative to each other.

Building CNN architecture

Now, that we have converted our dataset into images, we can start building our CNN architecture. But, first we need to split the data into train and test as we did in the previous example.

In [ ]:

```
from sklearn.model_selection import train_test_split
X=encoding(df)
X=np.array(X)
X.shape
Y=df["Tg"].values
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=42)
```

Now, we create a sequential model again just like how we build the architecture before. This problem is a regression task, which predicts the Glass Temperature of the polymer. Hence, while using the metric we use mean_absolute_error, Mean Absolute Error (MAE) is calculated by taking the summation of the absolute difference between the actual and calculated values of each observation over the entire array and then dividing the sum obtained by the number of observations in the array.

Here, we have also added a tuning-parameter learning rate, the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.[7]

Notice, in last layer we only have one neuron added this is because it's a regression task the output is only one value, i.e Glass temperature.

In [ ]:

```
model = Sequential()
model.add(Conv2D(64,(5,5), activation="relu",kernel_regularizer="l2",input_shape=(65,15,1)))
model.add(Conv2D(32,(3,3),kernel_regularizer="l2", activation="relu"))
model.add(MaxPool2D(pool_size=(3,3)))
model.add(Flatten())
model.add(Dense(32,activation="relu",kernel_regularizer="l2"))
model.add(Dense(10))
model.add(Dense(1))

from keras import optimizers
optimizer=optimizers.Adam(lr=0.003)

model.compile(optimizer=optimizer,loss="mean_absolute_error",)
```

Printing the model summary, to see the architecture.

In [ ]:

```
model.summary()
```

Training

Fitting the model to test and train dataset, this time we have significantly increased the epochs. As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to fitting curve sometimes might even lead to overfiiting.

In [ ]:

```
Model=model.fit(x=X_train,y=y_train,epochs=200)
```

Testing

In [ ]:

```
y_predtrain=model.predict(X_train)
y_predtest=model.predict(X_test)

MAE_test=abs(y_predtest.reshape(y_test.shape)-y_test).sum()/y_test.shape
MAE_train=abs(y_predtrain.reshape(y_train.shape)-y_train).sum()/y_train.shape

print("Mean Absolute Error on Training Set = ",MAE_train.item())
print("Mean Absolute Error on Test Set = ",MAE_test.item())
```

To understand the above output, consider example MAE=5 implies that, on average, the prediction's distance from the ground truth is 5 (e.g true value is 50 and prediction is 45 or ground truth is 50 and forecast is 55 would be a distance of 5). The closer MAE is to 0, the more accurate the model is. This is an acceptable model, but we can bring the MAE more down by adding more layers and using features like dropout, decreasing the learning rate and introducing more epochs.

Summary

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It’s also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image. CNN plays an important role towards the computer vision. It uses kernel for detecting features in the image. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. For identifying and recognizing objects, CNN architecture works really good.

Additional Resources:

In convolution operation, the arrays are multiplied element-wise, and the product is summed to create a new array, which represents a*b. Here, one can understand the convolution operation.

Here, we can visualize how a CNN network is actually working.

Bibliography

1. A Comprehensive Guide to Convolutional Neural Networks https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

2. Convolutional Neural Network Tutorial https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

3. Beginners Guide to Convolutional Neural Network with Implementation in Python https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/

4. Foundations of Convolutional Neural Networks https://www.coursera.org/learn/convolutional-neural-networks/home/week/1/

5. MIT 6.S191: Convolutional Neural Networks https://www.youtube.com/watch?v=uapdILWYTzE

6. Luis A. Miccio, Gustavo A. Schwartz, From chemical structure to quantitative polymer properties prediction through convolutional neural networks, Polymer, Volume 193, 2020, 122341, ISSN 0032-3861, https://doi.org/10.1016/j.polymer.2020.122341.

7. Learning rate https://en.wikipedia.org/wiki/Learning_rate#:~:text=In%20machine%20learning%20and%20statistics,minimum%20of%20a

In [ ]:

```
# Run this cell to generate a pdf from this notebook
# Click the generated links to preview and download it.
# Report errors to Professor Kitchin
from f22_06623 import pdf
%pdf
```

In [ ]:

```
```