DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 12:

HW No. 12: Convolutional Neural Networks

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

April 12th, 2023

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

A. TASK 1

The goal of this assignment is to explore convolutional neural networks and to adjust the network with varying frame size and max pooling layer. I do not have a particularly large foundation of knowledge regarding convolutional neural networks, so I would like to spend some time summarizing and exploring the concepts provided in the article given to us. Convolutional neural networks (CNNs) are a deep learning algorithm that are designed specifically for working with images or videos. These algorithms rely on images as inputs and functions by extracting and learning the features of these images to perform certain classifications.

An interesting aside that this article makes is that a CNN is inspired by the visual cortex (VC). This portion of the human brain works to process visual information from the outside world. The various layers of the VC each perform some function to extract visual information. This is quite similar to the layers that we will include in our CNN as the CNN has a variety of filters that also work to extract information (edges, shapes, colors, etc…).

A CNN model works in two steps, generally: feature extraction and classification. Feature extraction is the phase that contains filters and layers that are applied to the images in order to extract information from them. Once this phase is finished, it continues on to the classification phase. Here, the information is classified based on the target variable of the presented problem.

A typical CNN contains the following layers: input layer, convolutional layer, activation function, pooling layer, and a fully connected layer. The input layer is our initial image. The image can be in color or in grayscale (in our case, grayscale). At this point, it is important that we normalize the images and convert the range between 0 and 1. The frame sizes that we specify will determine the input matrix that will function as a representation of our input image. The convolution layer is when we apply our filter to our input image in an effort to extract features. The filter works by being applied multiple times and results in a feature map. The resulting feature map contains information that will be necessary for later use in our model. Once the feature map has been created, an activation function is applied in order to introduce nonlinearity. The pooling layer is applied *after* the convolutional layer and works to reduce the overall dimensions of the feature map we previously generated. The point of this portion is to not only preserve any important information but then to also reduce the overall computation time and cost. To conclude our CNN, we need to classify. The fully connected layer classifies our input image into some label. This layer connects the information that was extracted from the previous steps to the output layer and allows us to classify the input to a desired label.

The concepts behind a CNN are simple to understand but as this is my first time implementing one, I think it is important to include these details for later reference for my own work. Overall, the concepts seem to be quite straight-forward, and this article was incredibly helpful for me to better understand *how* to implement.

The homework requires us to vary the input frame size between 4x4, 6x6, and 8x8 and to vary the frame size used in the max-pooling layer between 2x2, 3x3, 4x4. Following the code in the tutorial was easy, but it took me a few minutes to implement a method to show the error and loss rates. To begin, I followed the initial steps and implemented the required libraries for this tutorial. The only one that I added myself was matplotlib in order to create a plot of error for ease of visualization. I have omitted this portion of code because of its self-explanatory nature.

I then followed the steps in order to create a model. As usual, we need to define our data as variables and begin the steps of the CNN. Once the data is defined, we normalize, add the convolutional layer and the activation function, then pass the data through the pooling layer and through the fully connected layer in

order to obtain our classified data. The code for this model closely follows the tutorial but with only slight variation in spacing and variable names for my own convenience and comfort.

```
#%% build model
#loading data
(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(X_train.shape)
print(X_test.shape)

#normalizing the pixel values
X_train=X_train/255
X_test=X_test/255

#defining model
model=Sequential()
#adding convolution layer
model.add(Conv2D(32,(2,2),
                 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(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'])
```

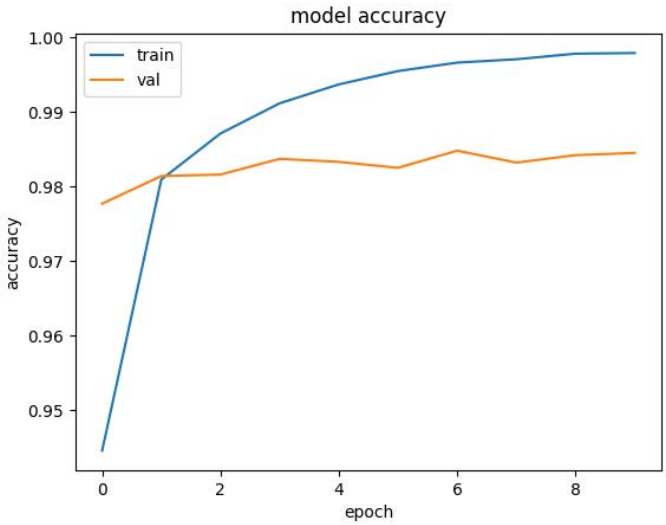
The first model that we run follows the tutorial exactly. Here we specify that the input shape is 28x28 and that the max pooling layer is 2x2. We maintain the choice of ReLU and softmax. We also know that we are working with a handwriting dataset. This dataset involves input images of only one number between 0 and 9. Thus, each input can only belong to one class and the labels are exclusive for each data point. This data works perfectly for sparse categorical cross entropy. Categorical cross entropy, on the other hand, is better used when the number of classes is larger than 2.

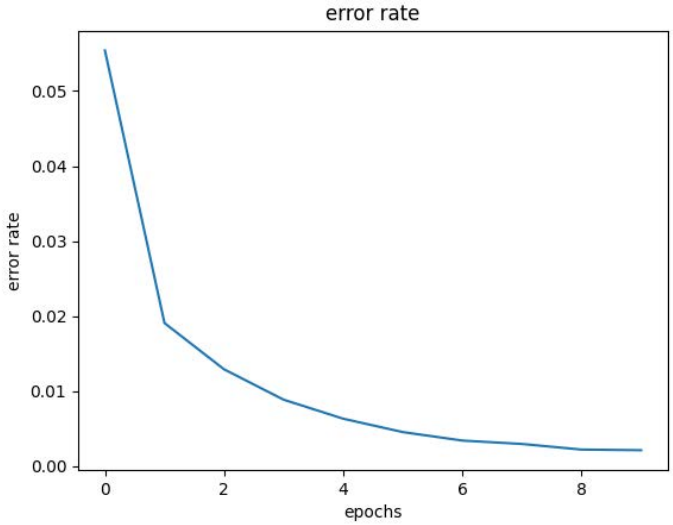
At this point, I alter the code in order to create a method to track the accuracy, loss, and error that occurs from our model. I specify a tuple variable that contains our test data and reference this in later code in order to produce a variety of plots. With an input kernel size of 2x2 and a max-pooling layer of 2x2, we are able to achieve an incredibly low error rate: 0.01591.

It is important to note that the loss rate & accuracy and the error rate needed to be plotted on separate graphs in order to efficiently see trends. The graphs produced are as follows and are produced with the following code:

```
#%%
#fitting the model
history = model.fit(X_train,y_train,
                    epochs=10,
                    verbose=1,
                    validation_data=(X_test,y_test))

#%%
error = []
for val in history.history['accuracy']:
    errorval = 1-val
    error.append(errorval)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.figure()
plt.plot(error)
plt.title("error rate")
plt.xlabel("epochs")
plt.ylabel("error rate")
plt.show()

#%% evaluate
model.evaluate(X_test,y_test)
```

Input Size (2x2) & Max-Pooling Layer (2x2)





B. CONCLUSION

To conclude this assignment, I have compiled all of the error rates from the 9 different parameter combination possibilities below.

Varying Input Layer and Max-Pooling Layer of CNN with 10 Epochs

Input Size	Max Pool Size	Error Rate
4x4	2x2	0.0106
4x4	3x3	0.0125
4x4	4x4	0.0115
6x6	2x2	0.0117
6x6	3x3	0.0121
6x6	4x4	0.0102
8x8	2x2	0.0117
8x8	3x3	0.0099
8x8	4x4	0.0105

The values of these parameters do not particularly change significantly between the varying combinations. However, the lowest error rate seen is found when the input size is 8x8 and the max-pooling layer size is 3x3; at this point, the error rate is only 0.0099.

I was curious though to determine why there was such little change between error rates. My initial assumption when completing this assignment was that there would be a more notable increase or decrease in error rate as these parameters were altered. To my confusion, they stayed relatively stagnant and so I wanted to spend some time researching the effects of the input-layer size and the max-pooling layer size.

The input 2D kernel size is the size of the convolutional filter that is used to extract any features from our images. If we increase the kernel size, we also increase the computational cost and the memory requirements of the network. Usually, we want to keep the kernel size small in order to minimize the cost and also to maintain sufficient enough resolution to obtain important details. In general, larger kernels can be used earlier in the network in order to increase the likelihood that we are extracting basic features of our images. When increasing the max-pooling size, we reduce the spatial resolution of the feature maps. This could be helpful when trying to improve the ability of our model to generalize and to reduce overfitting, but it also will reduce the model's capability to notice small fine-grained details in the provided image. For this reason, it is generally assumed that larger pooling sizes will be used in earlier network stages to extract coarse features while smaller pooling sizes will be used later to capture fine details.

With this information, I am gathering that a larger input size at the beginning of the network was important for us to be able to capture basic features of the handwritten numbers while a medium/small max-pooling layer was needed for us to capture the smaller and finer differences between numbers. Overall, this assignment was incredibly helpful because it made me dive deeper into the documentation regarding convolutional neural networks. Without doing so, I don't think I would have the same understanding of the separate steps required to make a CNN as I do now.