Project 1: Convolutional LSTM Neural Networks

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have achieved superb image classification capabilities due to their ability to learn spatial features in signals. As shown in Figure 1.1, a typical CNN architecture consists of an input (typically a two-dimensional signal such as an image), followed by one or more convolutional and max-pooling layers, one or more dense layers, and finally an output layer. In each convolutional layer, parameters of a two-dimensional filter are learned. These filters are then convolved with the input to produce various feature maps at each layer. The convolutional layer is usually followed by a maxpooling layer in which the highest value in a *pool* is retained and the other values are discarded. Typically, maxpooling layers specify 2×2 pool sizes where, among a 2×2 block of values, the highest value is passed, and the other three are discarded. This occurs in each 2×2 block in the feature map. The output of the maxpooling layer preceding the dense layers is typically flattened into vectors before they are passed into the dense layers. The dense layers, or fully connected layers, are vectors consisting of trainable parameters. The output, following the dense layer(s), is a fully connected softmax layer.

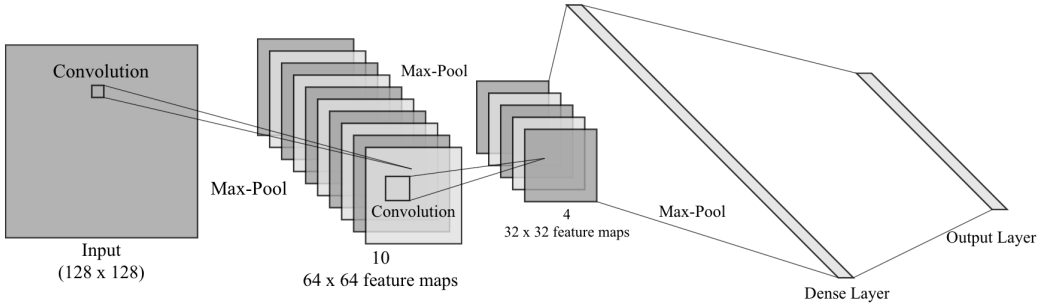

Figure 1.1: The architecture of a Convolutional Neural Network (CNN).

Similar to the robustness CNNs have achieved on images, long short-term-memory (LSTM) layers have shown to perform well on sequential data such as time-series data and natural language processing. Initially, LSTM layers were proposed to combat the vanishing (and exploding) gradient problem, but they were later found to be a good model for learning sequential data. LSTM uses a Recurrent Neural Network (RNN) architecture in which the current input to the layer consists of the output of the preceding layer as well as the output of the LSTM layer from the previous pass. The "memory" provided to the layer from its previous pass provides context for classification. This property allows LSTM models to learn trends in data that can be predicted from previous knowledge. For example, in a next-word text prediction algorithm, a much more accurate prediction can be made given the preceding words in the sentence. Convolutional layers and LSTM layers can be sequentially used in the same model, called a convolutional long-short-term deep neural network (CLDNN), to attain the effectiveness of convolution along with the benefits of LSTM layers. We will explore this architecture in this project.

The objective of this project is to implement a simple CNN to classify fashion articles from the Fashion-MNIST dataset (samples shown in Figure 1.2 below). We will also explore dropout and observe its effects when it is implemented in various layers of a CNN. Lastly, we will use our observations from the results of the Fashion-MNIST dataset classification to implement a CLDNN on time-series data for wireless RF modulation classification using DeepSig's RadioML Dataset. Both of the aforementioned datasets contain samples that belong to one of ten possible classes.

For this project, the Fashion-MNIST data can be loaded using Keras (code for this is provided in the given Jupyter Notebook) and the RadioML modulation classification data can be downloaded from DeepSig (the given Jupyter Notebook loads the data of interest from DeepSig and reshapes it appropriately). You are expected to fill in the Jupyter Notebook provided to you as instructed in each of the following sections. You may find it useful to refer to `https://keras.io` as you work through this and future projects in the course. **Submit your final code as a single .ipynb file. Ensure that the deliverables are clearly shown for each part to earn full credit. Name your submission *lastname_firstname_cldnn.ipynb.***

Figure 1.2: Example images from the Fashion-MNIST dataset.

2 EXAMPLE CNN IMPLEMENTATION

Below is an example of how to implement a basic CNN in Python using Keras and TensorFlow. Following the input layer, the CNN consists of a convolutional layer consisting of 64 feature maps and an 8×8 filter with 25% dropout, a 2×2 max-pooling layer, a layer to flatten the data into a vector, a dense layer of 256 perceptrons, and a softmax output layer.

Listing 1: CNNs in Keras

```
#ECE 595 Machine Learning II
#Project 1: CLDNN - Example Code
#Written by: Rajeev Sahay and Aly El Gamal

import numpy as np
from keras import Sequential
import keras
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt

num_samples = 5000
n = 10
num_channels = 3
num_classes = 12


#Create 5000 samples of random training data
data_train = np.random.rand(5000, n, n, num_channels)

#Create 1000 samples of random testing data
data_test = np.random.rand(1000, n, n, num_channels)

#Assign labels to training and testing data
labels_train = np.random.randint(low=0, high=11, size=(5000,))
labels_test = np.random.randint(low=0, high=11, size=(1000,))

#Transform labels to one-hot vectors
labels_train = keras.utils.np_utils.to_categorical(labels_train,
                                                   num_classes=num_classes)

labels_test = keras.utils.np_utils.to_categorical(labels_test,
                                                  num_classes=num_classes)


# Create a function that returns an initialized cnn model
def cnn():

    # Sequential allows model to be built layer by layer
    model = Sequential()

    # Bild a 2D Convolutional layer with 64 feature maps and a 8x8 filter
    # first layer requires a specified input shape
    model.add(Conv2D(64,
```

```
                        (8, 8),
                        activation='sigmoid',
                        input_shape=(n, n, num_channels)))

    # Randomly freezes 25% of weights in previous layer to prevent overfitting
    model.add(Dropout(0.25))

    # Add a 2x2 MaxPooling layer
    model.add(MaxPooling2D(pool_size=(2, 2),
                           strides=None,
                           padding='valid',
                           data_format=None))

    # Flatten data for input into Dense layers
    model.add(Flatten())

    # Add Dense layer with 256 perceptrons
    model.add(Dense(256,
                    activation="relu",
                    kernel_initializer="normal"))

    # Output layer with num_classes perceptrons
    model.add(Dense(num_classes,
                    activation="softmax"))

    return model

# Create graph of model
cnn_model = cnn()

# Compile the model
cnn_model.compile(loss='mean_squared_error',
                  optimizer='RMSprop',
                  metrics=['accuracy'])

# Train the model
cnn_model_mdata = cnn_model.fit(data_train, labels_train,
                                validation_data=(data_test, labels_test),
                                epochs=100,
                                batch_size=1024,
                                shuffle=True)


#Evaluate the accuracy of the model on the testing set
scores = cnn_model.evaluate(data_test, labels_test)

#Print the accuracy of the testing set
print("Accuracy: %.2f%%" %(scores[1]*100))

# Plot the accuracy vs epoch
```

```
94  plt.plot(cnn_model_mdata.history['acc'])
95  plt.plot(cnn_model_mdata.history['val_acc'])
96  plt.title('CNN Accuracy vs. Epoch')
97  plt.ylabel('Accuracy')
98  plt.xlabel('Epoch')
99  plt.legend(['train', 'test'], loc='upper left')
100 plt.show()
101
102 # Plot the loss vs epoch
103 plt.plot(cnn_model_mdata.history['loss'])
104 plt.plot(cnn_model_mdata.history['val_loss'])
105 plt.title('CNN Loss vs. Epoch')
106 plt.ylabel('loss')
107 plt.xlabel('epoch')
108 plt.legend(['train', 'test'], loc='upper left')
109 plt.show()
```

3 PART 1: LOSS FUNCTION VISUALIZATION

Several loss functions are available to use as the objective function when optimizing neural networks such as the mean squared error and the negative log-likelihood. Each loss function can be valuable for different types of architectures. In the models that will be developed in this project, the output layer will use a softmax activation, which will allow each of the output classes to be interpreted as probabilities. The negative log-likelihood loss, or categorical cross entropy, is thus a good loss function to use. In this section, you are to obtain and analyze the plot of the negative log-likelihood function.

Deliverables for Part 1:

1. Show a plot of the negative log-likelihood vs. the probability, p, for $p \in [0, 1]$. Include appropriate axes values, axes labels, and a title.
2. Answer the following questions:
    a) How is the the negative log-likelihood of p affected when p is small? How about when p is large?
    b) Why does the behavior of the negative log-likelihood function make it a good objective function for a minimization problem specifically when the output can be interpreted as a probability space?

4 PART 2: OVERFIT CNN

Implement a CNN to classify fashion articles from the Fashion-MNIST dataset according to the following specifications:

- Input layer: a $28 \times 28 \times 1$ image (the 1 specifies a grayscale image)
- First hidden layer: a 2-dimensional convolutional layer with 256 feature maps and a 3×3 filter size
- Second hidden layer: a 2×2 max-pooling layer
- Third hidden layer: a 2-dimensional convolutional layer with 128 feature maps and a 3×3 filter size
- Fourth hidden layer: a 2×2 max-pooling layer
- Fifth hidden layer: a layer to *flatten* the data
- Sixth hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Seventh hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Output layer (classification probabilites): 10 perceptrons
- Train your model using **200 epochs**

You will have to make decisions about which activation functions to use at each layer, whether or not to use a bias for each layer, which optimizer to use for updating the parameter values during training, and the batch size. Use softmax as the output layer. Since output layer is fixed as softmax, choose loss function that best works with it.

Deliverables for Part 2:

1. Print the accuracy of the trained model on the testing set
2. Show a plot of the accuracy versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
3. Show a plot of the model loss versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
4. Answer the following questions:

    a) What does the loss of the training set go to?
    b) What does the loss of the testing set go to?
    c) What is the reason for the discrepancy between the training and testing set loss?
    d) Explain why the accuracy of the testing set, after training, differs so much from the testing set regardless of achieving high training accuracy. Name two ways to avoid this.

5 Part 3: Input Layer Dropout

Implement a CNN to classify fashion articles from the Fashion-MNIST dataset according to the following specifications (Note: The architecture here is identical to that in Part 2 except with a 30% dropout rate applied to the input layer):

- Input layer: a $28 \times 28 \times 1$ image with a 30% dropout rate
- First hidden layer: a 2-dimensional convolutional layer with 256 feature maps and a 3×3 filter size
- Second hidden layer: a 2×2 max-pooling layer
- Third hidden layer: a 2-dimensional convolutional layer with 128 feature maps and a 3×3 filter size
- Fourth hidden layer: a 2×2 max-pooling layer
- Fifth hidden layer: a layer to *flatten* the data
- Sixth hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Seventh hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Output layer (classification probabilites): 10 perceptrons
- Train your model using **200 epochs**

Deliverables for Part 3:

1. Print the accuracy of the trained model on the testing set
2. Show a plot of the accuracy versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
3. Show a plot of the model loss versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
4. Answer the following questions:

    a) What does the loss of the training set go to?
    b) What does the loss of the testing set go to?
    c) Why is the loss of the training set so different from the loss of the testing set regardless of using dropout?

6 Part 4: Hidden Layer Dropout

Implement a CNN to classify fashion articles from the Fashion-MNIST dataset according to the following specifications (Note: The architecture here is identical to that in Part 2 except with a 30% dropout rate applied to the first and second convolutional layers):

- Input layer: a $28 \times 28 \times 1$ image
- First hidden layer: a 2-dimensional convolutional layer with 256 feature maps and a 3×3 filter size with a 30% dropout rate
- Second hidden layer: a 2×2 max-pooling layer
- Third hidden layer: a 2-dimensional convolutional layer with 128 feature maps and a 3×3 filter size with a 30% dropout rate
- Fourth hidden layer: a 2×2 max-pooling layer
- Fifth hidden layer: a layer to *flatten* the data
- Sixth hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Seventh hidden layer: A dense (fully-connected) layer consisting of 100 perceptrons
- Output layer (classification probabilites): 10 perceptrons
- Train your model using **200 epochs**

Deliverables for Part 4:

1. Print the accuracy of the trained model on the testing set
2. Show a plot of the accuracy versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
3. Show a plot of the model loss versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
4. Answer the following questions:
    a) What does the loss of the training set go to?
    b) What does the loss of the testing set go to?
    c) Why was using dropout more effective here in comparison to the architecture in Part 3?
    d) What is the difference in accuracy, after training, between the training and testing set? Is the model a good fit for the data?
5. Keep the same network and Add dropout layer between the Input layer and the First hidden layer with 30% dropout rate. [Multi-layer Dropout]
    a) What does the loss of the training set go to?
    b) What does the loss of the testing set go to?
    c) What is the difference in accuracy, after training, between the training and testing sets?
    d) Compare the results of 4 above with the obtained results in 5. Comment on robustness and accuracy.

7 PART 5: ACTIVATION FUNCTIONS

Implement a CNN to classify fashion articles from the Fashion-MNIST dataset according to the following specifications (Note: The architecture here is identical to that in Part 4, 4^{th} question):

With the same hyper-parameters from the previous part, assess the effects of different output layer activations on the training. Keep the loss function fixed as the cross-entropy loss.

Deliverables for Part 5:

1. Sigmoid activation: Use the 'sigmoid' activation function for the output layer and compare the following with softmax results.
    a) Show a plot of the accuracy versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
    b) Show a plot of the model loss versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
    c) Compare the above results with softmax results [with point 5 results of Part 4]
2. Tanh activation: Use the 'tanh' activation function for the output layer.
    a) Does 'tanh' as output activation work with Cross-entropy loss? If yes, report the accuracy plot for both training and testing. If not, try MSE loss and report the results.
    b) Give a reason why tanh is not recommended with cross entropy loss?
    c) Comment on why sigmoid does not cause the same issues as tanh when used with Cross entropy loss?
3. Bonus:
    a) Change hidden units to sigmoid and tanh and report the accuracy and loss plots for the training and testing sets.
    b) When using ReLU hidden layer activations, compare average activity [number of active neurons] in layers during training and testing.
        i. Note 1: Choose and fix few images over which you want to calculate the average activity in the layers during training and testing.
        ii. Note 2: Choose certain epochs on which you will be calculating the activity. Doing it on every epoch will reduce your speed of training.

8 PART 6: IMPLEMENTING A CLDNN

Implement a CLDNN to classify the modulation of time-series data as one of ten possible modulation types. The given code file imports the high SNR data, which is already split into a training and testing set. Ensure that the data files accompanying this project (X_train.npy, Y_train.npy, X_test.npy, and Y_test.npy) are in the same directory as your Python script. Build your model according to the following architecture:

- Input layer: a $2 \times 128 \times 1$ signal
- First hidden layer: a 2-dimensional convolutional layer with 256 feature maps and a 1×3 filter size with a 20% dropout rate
- Second hidden layer: a 2-dimensional convolutional layer with 256 feature maps and a 2×3 filter size
- Third hidden layer: a 2-dimensional convolutional layer with 80 feature maps and a 1×3 filter size with a 20% dropout rate
- Fourth hidden layer: a 2-dimensional convolutional layer with 80 feature maps and a 1×3 filter size
- Reshape the data to be two-dimensional instead of three-dimensional
- Fifth hidden layer: An LSTM layer with 50 cells
- Sixth hidden layer: a dense (fully-connected) layer consisting of 128 perceptrons
- Output layer (classification probabilites): 10 perceptrons
- Train your model using **200 epochs**

You will have to make decisions about which activation functions to use at each layer, whether or not to use a bias for each layer, which loss function to use for training, which optimizer to use for updating the parameter values during training, and the batch size.

Deliverables for Part 6:

1. Print the accuracy of the trained model on the testing set
2. Show a plot of the accuracy versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
3. Show a plot of the model loss versus the epochs for both the training and validation data (the validation data here can be the testing data). Include a legend, appropriate axes labels, and a title.
4. Answer the following questions:

    a) Is this model a good fit for the data?
    b) Give two ways to improve the robustness of the model.