task2

May 15, 2024

1 Task 2

A Convolutional Neural Network (CNN) is a specialized type of deep neural network designed for processing grid-like data structures, most commonly images. A CNN consists of several key components: an input layer that receives the raw data, convolutional layers that apply filters to detect local patterns, activation functions like ReLU to introduce non-linearity, and pooling layers to down-sample the feature maps, reducing their dimensionality while retaining important information. After several convolutional and pooling operations, fully connected layers are used to perform high-level reasoning, culminating in an output layer that produces the final predictions, often using a softmax activation for classification tasks. CNNs leverage local receptive fields and parameter sharing, which make them highly effective at learning hierarchical features. These networks have revolutionized the field of computer vision and are widely applied in tasks such as image recognition, object detection, and, increasingly, in cybersecurity for applications like network intrusion detection.

[22]:
```python
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# For demonstration, we create a random dataset
np.random.seed(42)
data_size = 1000
feature_size = 20
data = pd.DataFrame(np.random.rand(data_size, feature_size),␣
 ↪columns=[f'feature_{i}' for i in range(feature_size)])
data['label'] = np.random.randint(0, 2, size=data_size)

# Output 10 rows from the generated data
print("Sample data (10 rows):")
print(data.head(10))

# Reshape data for CNN input (e.g., treat features as 1D spatial data)
X = data.drop('label', axis=1).values.reshape(-1, feature_size, 1)
y = data['label']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.reshape(-1, feature_size)).reshape(-1,␣
↪feature_size, 1)
X_test = scaler.transform(X_test.reshape(-1, feature_size)).reshape(-1,␣
↪feature_size, 1)

# Define the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu',␣
↪input_shape=(feature_size, 1)),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,␣
↪validation_split=0.2)

# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f'CNN Accuracy: {accuracy * 100:.2f}%')
```

```
Sample data (10 rows):
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.374540   0.950714   0.731994   0.598658   0.156019   0.155995
1   0.611853   0.139494   0.292145   0.366362   0.456070   0.785176
2   0.122038   0.495177   0.034389   0.909320   0.258780   0.662522
3   0.388677   0.271349   0.828738   0.356753   0.280935   0.542696
4   0.863103   0.623298   0.330898   0.063558   0.310982   0.325183
5   0.031429   0.636410   0.314356   0.508571   0.907566   0.249292
6   0.807440   0.896091   0.318003   0.110052   0.227935   0.427108
7   0.962447   0.251782   0.497249   0.300878   0.284840   0.036887
```

```
8   0.367783   0.632306   0.633530   0.535775   0.090290   0.835302
9   0.341066   0.113474   0.924694   0.877339   0.257942   0.659984

   feature_6  feature_7  feature_8  feature_9  …  feature_11  feature_12  \
0   0.058084   0.866176   0.601115   0.708073  …    0.969910    0.832443
1   0.199674   0.514234   0.592415   0.046450  …    0.170524    0.065052
2   0.311711   0.520068   0.546710   0.184854  …    0.775133    0.939499
3   0.140924   0.802197   0.074551   0.986887  …    0.198716    0.005522
4   0.729606   0.637557   0.887213   0.472215  …    0.713245    0.760785
5   0.410383   0.755551   0.228798   0.076980  …    0.161221    0.929698
6   0.818015   0.860731   0.006952   0.510747  …    0.222108    0.119865
7   0.609564   0.502679   0.051479   0.278646  …    0.239562    0.144895
8   0.320780   0.186519   0.040775   0.590893  …    0.016588    0.512093
9   0.817222   0.555201   0.529651   0.241852  …    0.897216    0.900418

   feature_13  feature_14  feature_15  feature_16  feature_17  feature_18  \
0    0.212339    0.181825    0.183405    0.304242    0.524756    0.431945
1    0.948886    0.965632    0.808397    0.304614    0.097672    0.684233
2    0.894827    0.597900    0.921874    0.088493    0.195983    0.045227
3    0.815461    0.706857    0.729007    0.771270    0.074045    0.358466
4    0.561277    0.770967    0.493796    0.522733    0.427541    0.025419
5    0.808120    0.633404    0.871461    0.803672    0.186570    0.892559
6    0.337615    0.942910    0.323203    0.518791    0.703019    0.363630
7    0.489453    0.985650    0.242055    0.672136    0.761620    0.237638
8    0.226496    0.645173    0.174366    0.690938    0.386735    0.936730
9    0.633101    0.339030    0.349210    0.725956    0.897110    0.887086

   feature_19  label
0    0.291229      1
1    0.440152      1
2    0.325330      1
3    0.115869      0
4    0.107891      0
5    0.539342      0
6    0.971782      1
7    0.728216      0
8    0.137521      1
9    0.779876      0

[10 rows x 21 columns]
Epoch 1/50

/opt/anaconda3/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

```
18/18              0s 4ms/step -
accuracy: 0.5397 - loss: 0.7015 - val_accuracy: 0.5000 - val_loss: 0.6912
Epoch 2/50
18/18              0s 1ms/step -
accuracy: 0.5376 - loss: 0.6896 - val_accuracy: 0.5143 - val_loss: 0.6937
Epoch 3/50
18/18              0s 1ms/step -
accuracy: 0.5478 - loss: 0.6859 - val_accuracy: 0.5857 - val_loss: 0.6872
Epoch 4/50
18/18              0s 1ms/step -
accuracy: 0.5673 - loss: 0.6834 - val_accuracy: 0.5000 - val_loss: 0.6911
Epoch 5/50
18/18              0s 1ms/step -
accuracy: 0.5908 - loss: 0.6769 - val_accuracy: 0.5143 - val_loss: 0.6866
Epoch 6/50
18/18              0s 2ms/step -
accuracy: 0.6049 - loss: 0.6740 - val_accuracy: 0.5071 - val_loss: 0.6918
Epoch 7/50
18/18              0s 2ms/step -
accuracy: 0.6758 - loss: 0.6415 - val_accuracy: 0.5714 - val_loss: 0.6862
Epoch 8/50
18/18              0s 2ms/step -
accuracy: 0.6460 - loss: 0.6610 - val_accuracy: 0.4786 - val_loss: 0.6901
Epoch 9/50
18/18              0s 2ms/step -
accuracy: 0.6570 - loss: 0.6457 - val_accuracy: 0.5500 - val_loss: 0.6822
Epoch 10/50
18/18              0s 2ms/step -
accuracy: 0.6680 - loss: 0.6225 - val_accuracy: 0.5214 - val_loss: 0.6828
Epoch 11/50
18/18              0s 2ms/step -
accuracy: 0.6970 - loss: 0.6161 - val_accuracy: 0.5286 - val_loss: 0.6812
Epoch 12/50
18/18              0s 2ms/step -
accuracy: 0.6983 - loss: 0.6092 - val_accuracy: 0.5143 - val_loss: 0.6870
Epoch 13/50
18/18              0s 2ms/step -
accuracy: 0.6868 - loss: 0.6083 - val_accuracy: 0.5429 - val_loss: 0.6850
Epoch 14/50
18/18              0s 2ms/step -
accuracy: 0.7549 - loss: 0.5723 - val_accuracy: 0.5571 - val_loss: 0.6927
Epoch 15/50
18/18              0s 2ms/step -
accuracy: 0.7857 - loss: 0.5232 - val_accuracy: 0.5357 - val_loss: 0.6948
Epoch 16/50
18/18              0s 2ms/step -
accuracy: 0.7662 - loss: 0.5215 - val_accuracy: 0.5429 - val_loss: 0.6849
Epoch 17/50
```

```
18/18            0s 2ms/step -
accuracy: 0.7505 - loss: 0.5093 - val_accuracy: 0.5786 - val_loss: 0.7483
Epoch 18/50
18/18            0s 2ms/step -
accuracy: 0.7381 - loss: 0.5152 - val_accuracy: 0.5643 - val_loss: 0.7401
Epoch 19/50
18/18            0s 2ms/step -
accuracy: 0.7753 - loss: 0.4731 - val_accuracy: 0.5571 - val_loss: 0.7514
Epoch 20/50
18/18            0s 2ms/step -
accuracy: 0.8493 - loss: 0.4204 - val_accuracy: 0.5429 - val_loss: 0.7458
Epoch 21/50
18/18            0s 2ms/step -
accuracy: 0.8133 - loss: 0.4110 - val_accuracy: 0.5571 - val_loss: 0.7762
Epoch 22/50
18/18            0s 2ms/step -
accuracy: 0.8537 - loss: 0.3983 - val_accuracy: 0.5786 - val_loss: 0.7576
Epoch 23/50
18/18            0s 2ms/step -
accuracy: 0.8727 - loss: 0.3469 - val_accuracy: 0.5786 - val_loss: 0.7973
Epoch 24/50
18/18            0s 2ms/step -
accuracy: 0.8843 - loss: 0.3217 - val_accuracy: 0.5929 - val_loss: 0.7690
Epoch 25/50
18/18            0s 2ms/step -
accuracy: 0.8592 - loss: 0.3199 - val_accuracy: 0.5500 - val_loss: 0.8273
Epoch 26/50
18/18            0s 2ms/step -
accuracy: 0.8873 - loss: 0.2900 - val_accuracy: 0.5786 - val_loss: 0.8609
Epoch 27/50
18/18            0s 2ms/step -
accuracy: 0.9179 - loss: 0.2637 - val_accuracy: 0.5929 - val_loss: 0.8662
Epoch 28/50
18/18            0s 1ms/step -
accuracy: 0.8999 - loss: 0.2575 - val_accuracy: 0.6143 - val_loss: 0.8987
Epoch 29/50
18/18            0s 1ms/step -
accuracy: 0.9468 - loss: 0.2157 - val_accuracy: 0.5929 - val_loss: 0.9271
Epoch 30/50
18/18            0s 1ms/step -
accuracy: 0.9173 - loss: 0.2261 - val_accuracy: 0.5643 - val_loss: 0.9749
Epoch 31/50
18/18            0s 2ms/step -
accuracy: 0.9547 - loss: 0.1907 - val_accuracy: 0.5714 - val_loss: 1.0233
Epoch 32/50
18/18            0s 1ms/step -
accuracy: 0.9731 - loss: 0.1521 - val_accuracy: 0.5857 - val_loss: 1.0271
Epoch 33/50
```

```
18/18             0s 1ms/step -
accuracy: 0.9705 - loss: 0.1344 - val_accuracy: 0.5857 - val_loss: 1.1377
Epoch 34/50
18/18             0s 2ms/step -
accuracy: 0.9586 - loss: 0.1403 - val_accuracy: 0.5643 - val_loss: 1.1056
Epoch 35/50
18/18             0s 1ms/step -
accuracy: 0.9688 - loss: 0.1167 - val_accuracy: 0.5643 - val_loss: 1.1772
Epoch 36/50
18/18             0s 2ms/step -
accuracy: 0.9713 - loss: 0.1096 - val_accuracy: 0.5786 - val_loss: 1.1556
Epoch 37/50
18/18             0s 1ms/step -
accuracy: 0.9825 - loss: 0.0929 - val_accuracy: 0.5857 - val_loss: 1.2567
Epoch 38/50
18/18             0s 1ms/step -
accuracy: 0.9626 - loss: 0.0987 - val_accuracy: 0.6071 - val_loss: 1.2027
Epoch 39/50
18/18             0s 1ms/step -
accuracy: 0.9867 - loss: 0.0718 - val_accuracy: 0.5786 - val_loss: 1.2234
Epoch 40/50
18/18             0s 1ms/step -
accuracy: 0.9968 - loss: 0.0514 - val_accuracy: 0.6000 - val_loss: 1.2829
Epoch 41/50
18/18             0s 2ms/step -
accuracy: 0.9867 - loss: 0.0761 - val_accuracy: 0.5857 - val_loss: 1.3499
Epoch 42/50
18/18             0s 2ms/step -
accuracy: 0.9972 - loss: 0.0467 - val_accuracy: 0.5786 - val_loss: 1.3762
Epoch 43/50
18/18             0s 2ms/step -
accuracy: 0.9838 - loss: 0.0578 - val_accuracy: 0.5786 - val_loss: 1.3635
Epoch 44/50
18/18             0s 2ms/step -
accuracy: 0.9797 - loss: 0.0667 - val_accuracy: 0.6071 - val_loss: 1.4756
Epoch 45/50
18/18             0s 2ms/step -
accuracy: 0.9887 - loss: 0.0543 - val_accuracy: 0.6071 - val_loss: 1.4191
Epoch 46/50
18/18             0s 2ms/step -
accuracy: 0.9884 - loss: 0.0554 - val_accuracy: 0.5714 - val_loss: 1.4520
Epoch 47/50
18/18             0s 2ms/step -
accuracy: 0.9842 - loss: 0.0554 - val_accuracy: 0.5357 - val_loss: 1.5236
Epoch 48/50
18/18             0s 2ms/step -
accuracy: 0.9925 - loss: 0.0406 - val_accuracy: 0.5643 - val_loss: 1.5168
Epoch 49/50
```

```
18/18             0s 2ms/step -
accuracy: 1.0000 - loss: 0.0235 - val_accuracy: 0.5714 - val_loss: 1.5277
Epoch 50/50
18/18             0s 2ms/step -
accuracy: 0.9995 - loss: 0.0223 - val_accuracy: 0.5857 - val_loss: 1.5626
10/10             0s 2ms/step
CNN Accuracy: 51.33%
```

Please use the "Kernel>Restart & Run All" command in Jupyter Notebook and check your results before submitting your homework. Note that I rerun all boxes on my side before grading.