CNN vs Dense Layers

In the proposed approach, we employ a Dense to handle the code metrics. We hava also tried to replace it with CNN layers, and the resulting approach is presented in Fig.1. We call it CNN-based approach, and call the original one(introduced in the journal paper) as Dense-based approach.

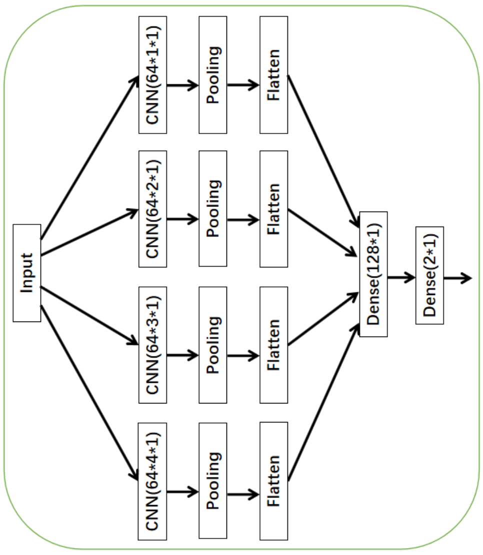

Figure 1. Replacing Dense with CNN Layer

The source code of the algorithm is presented as follows:

```
conv1=Sequential()
conv2=Sequential()
conv3=Sequential()
conv4=Sequential()

conv1.add(Conv1D(64,1,padding="same",input_shape=(9,1),activation="relu"))
conv1.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv1.add(Conv1D(64,1,padding="same",activation="relu"))
conv1.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv1.add(Conv1D(64,1,padding="same",activation="relu"))
conv1.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv1.add(Flatten())

conv2.add(Conv1D(64,2,padding="same",input_shape=(9,1),activation="relu"))
conv2.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv2.add(Conv1D(64,2,padding="same",activation="relu"))
conv2.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv2.add(Conv1D(64,2,padding="same",activation="relu"))
conv2.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
```

```
conv2.add(Flatten())

conv3.add(Conv1D(64,3,padding="same",input_shape=(9,1),activation="relu"))
conv3.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv3.add(Conv1D(64,3,padding="same",activation="relu"))
conv3.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv3.add(Conv1D(64,3,padding="same",activation="relu"))
conv3.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv3.add(Flatten())

conv4.add(Conv1D(64,4,padding="same",input_shape=(9,1),activation="relu"))
conv4.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv4.add(Conv1D(64,4,padding="same",activation="relu"))
conv4.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv4.add(Conv1D(64,4,padding="same",activation="relu"))
conv4.add(pooling.MaxPooling1D(pool_size=2,padding="same"))
conv4.add(Flatten())

output=merge.Concatenate()([conv1.output,conv2.output,conv3.output,conv4.output])
output=Dense(128, activation='relu')(output)
output=Dense(2,activation='sigmoid')(output)

input1=conv1.input
input2=conv2.input
input3=conv3.input
input4=conv4.input
model=Model([input1,input2,input3,input4],output)
model.compile(loss='binary_crossentropy',optimizer='Adadelta',metrics=['accuracy'])
```

With the algorithm, we repeat the evaluation and results are presented on Table 2. From this table, we observe the resulting precision and recall is both slightly smaller than the ones of the Dense-based approach (as presented in Table 1). In summary, the performance of the dense-based approach is better than that of the CNN-based approach by comparing the F1 value. Consequently, we keep the original Dense-based approach.

Table 1: Result of Dense-based Approach

Applications	Precision	Recall	F1
Areca	33.89%	80.17%	47.64%
Freeplane	36.72%	75.77%	49.47%
jEdit	36.95%	72.79%	49.02%
JExcelAPI	19.74%	62.72%	30.03%
JUnit	45.30%	46.49%	45.89%
PMD	27.92%	74.57%	40.63%
Weka	33.13%	78.82%	46.65%
Average	33.80%	75.91%	46.77%

Table 2: Result of CNN-based Approach

Applications	Precision	Recall	F1
Areca	31.66%	86.61%	46.37%
Freeplane	36.6%	70.96%	48.3%
jEdit	36.43%	79.38%	49.94%
JExcelAPI	17.94%	55.62%	27.13%
JUnit	43.36%	54.39%	48.25%
PMD	27.71%	82.66%	41.51%
Weka	32.67%	73.3%	45.2%
Average	33.14%	74.91%	45.95%