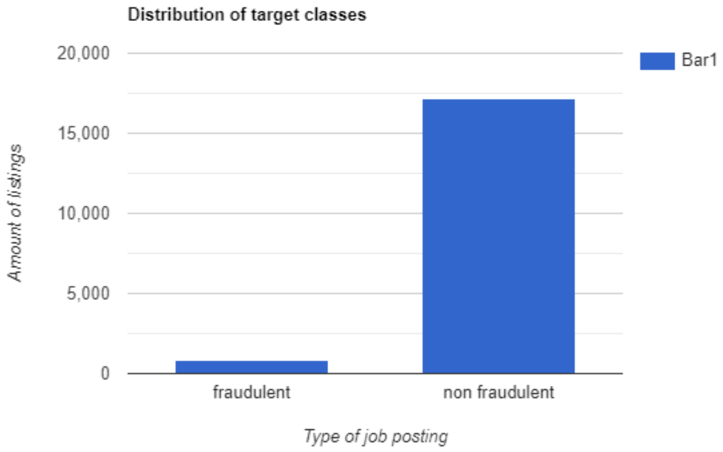
Text Classification 2

For our text classification, we will be looking at a data set that contains data of real and fraudulent job postings. The graph below shows the distribution of the data- 800 fraudulent postings and 17200 real postings.



First, we will preprocess the data and split into train and test.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from tensorflow.keras import datasets, layers, models, preprocessing


from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd

np.random.seed(1234)


df = pd.read_csv('../content/joblistings.csv', header=0, usecols=[6,17], encoding='latin-1',engine='python', error_bad_lines=False)
i = np.random.rand(len(df)) < 0.7
train = df[i]
test = df[~i]
num_labels = 2
vocab_size = 25000
batch_size = 100

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.description)

x_train = tokenizer.texts_to_matrix(train.description, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.description, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.fraudulent)
y_train = encoder.transform(train.fraudulent)
y_test = encoder.transform(test.fraudulent)
```

Lets try a sequential model on this test data

```
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 2/30
84/84 [==============================] - 3s 31ms/step - loss: 0.0777 - accuracy: 0.9749 - val_loss: 0.0763 - val_accuracy: 0.9849
Epoch 3/30
84/84 [==============================] - 3s 30ms/step - loss: 0.0329 - accuracy: 0.9905 - val_loss: 0.0782 - val_accuracy: 0.9849
Epoch 4/30
84/84 [==============================] - 3s 36ms/step - loss: 0.0141 - accuracy: 0.9976 - val_loss: 0.0884 - val_accuracy: 0.9860
Epoch 5/30
84/84 [==============================] - 3s 40ms/step - loss: 0.0083 - accuracy: 0.9990 - val_loss: 0.0926 - val_accuracy: 0.9849
Epoch 6/30
84/84 [==============================] - 2s 28ms/step - loss: 0.0054 - accuracy: 0.9993 - val_loss: 0.1072 - val_accuracy: 0.9871
Epoch 7/30
84/84 [==============================] - 2s 27ms/step - loss: 0.0050 - accuracy: 0.9994 - val_loss: 0.1119 - val_accuracy: 0.9860
Epoch 8/30
84/84 [==============================] - 3s 30ms/step - loss: 0.0039 - accuracy: 0.9994 - val_loss: 0.1098 - val_accuracy: 0.9860
Epoch 9/30
84/84 [==============================] - 3s 37ms/step - loss: 0.0038 - accuracy: 0.9994 - val_loss: 0.1130 - val_accuracy: 0.9860
Epoch 10/30
84/84 [==============================] - 3s 37ms/step - loss: 0.0021 - accuracy: 0.9995 - val_loss: 0.1219 - val_accuracy: 0.9860
Epoch 11/30
84/84 [==============================] - 3s 31ms/step - loss: 0.0032 - accuracy: 0.9994 - val_loss: 0.1250 - val_accuracy: 0.9860
Epoch 12/30
84/84 [==============================] - 3s 32ms/step - loss: 0.0025 - accuracy: 0.9995 - val_loss: 0.1227 - val_accuracy: 0.9860
Epoch 13/30
84/84 [==============================] - 2s 28ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.1219 - val_accuracy: 0.9871
Epoch 14/30
84/84 [==============================] - 3s 33ms/step - loss: 0.0020 - accuracy: 0.9996 - val_loss: 0.1300 - val_accuracy: 0.9860
Epoch 15/30
84/84 [==============================] - 3s 30ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.1318 - val_accuracy: 0.9860
Epoch 16/30
84/84 [==============================] - 2s 25ms/step - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.1287 - val_accuracy: 0.9871
Epoch 17/30
84/84 [==============================] - 3s 30ms/step - loss: 0.0017 - accuracy: 0.9998 - val_loss: 0.1399 - val_accuracy: 0.9860
Epoch 18/30
84/84 [==============================] - 2s 29ms/step - loss: 0.0010 - accuracy: 0.9998 - val_loss: 0.1306 - val_accuracy: 0.9871
Epoch 19/30
84/84 [==============================] - 3s 35ms/step - loss: 0.0017 - accuracy: 0.9998 - val_loss: 0.1467 - val_accuracy: 0.9860
Epoch 20/30
84/84 [==============================] - 3s 34ms/step - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.1375 - val_accuracy: 0.9860
Epoch 21/30
84/84 [==============================] - 2s 28ms/step - loss: 0.0010 - accuracy: 0.9998 - val_loss: 0.1349 - val_accuracy: 0.9871
Epoch 22/30
84/84 [==============================] - 2s 27ms/step - loss: 0.0013 - accuracy: 0.9999 - val_loss: 0.1502 - val_accuracy: 0.9860
Epoch 23/30
84/84 [==============================] - 2s 27ms/step - loss: 0.0016 - accuracy: 0.9998 - val_loss: 0.1405 - val_accuracy: 0.9871
Epoch 24/30
84/84 [==============================] - 2s 24ms/step - loss: 0.0013 - accuracy: 0.9998 - val_loss: 0.1506 - val_accuracy: 0.9860
Epoch 25/30
84/84 [==============================] - 3s 36ms/step - loss: 9.4716e-04 - accuracy: 0.9999 - val_loss: 0.1394 - val_accuracy: 0.9871
Epoch 26/30
84/84 [==============================] - 2s 27ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.1547 - val_accuracy: 0.9860
Epoch 27/30
84/84 [==============================] - 2s 30ms/step - loss: 9.1141e-04 - accuracy: 0.9999 - val_loss: 0.1446 - val_accuracy: 0.9871
Epoch 28/30
84/84 [==============================] - 2s 27ms/step - loss: 7.3394e-04 - accuracy: 0.9999 - val_loss: 0.1564 - val_accuracy: 0.9860
Epoch 29/30
84/84 [==============================] - 2s 25ms/step - loss: 7.1334e-04 - accuracy: 0.9999 - val_loss: 0.1483 - val_accuracy: 0.9860
Epoch 30/30
```

```
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
24/24 [==============================] - 0s 14ms/step - loss: 0.2094 - accuracy: 0.9809
Accuracy:  0.9809358716011047
```

```
print(score)
```

```
[0.20940755307674408, 0.9809358716011047]
```

```
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
```

```
73/73 [==============================] - 1s 8ms/step
```

```
pred[:10]
```

```
array([[1.1930392e-19],
       [3.4745897e-03],
       [8.9375745e-14],
       [9.5152354e-06],
       [2.6130313e-02],
       [1.6278232e-15],
       [7.0342692e-05],
       [3.2189888e-07],
       [5.4108055e-12],
       [4.1028156e-07]], dtype=float32)
```

```
pred_labels[:10]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
accuracy score:  0.9809358752166378
precision score:  0.9298245614035088
recall score:  0.5698924731182796
f1 score:  0.7066666666666666
```

The results are pretty good! The precision score is .92, meaning that a good amount of the fraudulent listings were actually labeled as such. Next we will try RNN.

```
max_features = 10000
maxlen = 500
batch_size = 32

# pad the data to maxlen
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

print("train shapes:", train_data.shape, y_train.shape)
print("test shapes:", test_data.shape, y_test.shape)
```

```
train shapes: (8126, 500) (8126,)
test shapes: (3464, 500) (3464,)
```

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          320000    
                                                                 
 simple_rnn_2 (SimpleRNN)    (None, 32)                2080      
                                                                 
 dense_8 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_data,
history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
51/51 [==============================] - 15s 262ms/step - loss: 0.2953 - accuracy: 0.9171 - val_loss: 0.1304 - val_accuracy: 0.9760
Epoch 2/10
51/51 [==============================] - 14s 277ms/step - loss: 0.1906 - accuracy: 0.9529 - val_loss: 0.1182 - val_accuracy: 0.9760
Epoch 3/10
51/51 [==============================] - 14s 279ms/step - loss: 0.1905 - accuracy: 0.9529 - val_loss: 0.1154 - val_accuracy: 0.9760
Epoch 4/10
51/51 [==============================] - 14s 270ms/step - loss: 0.1908 - accuracy: 0.9529 - val_loss: 0.1164 - val_accuracy: 0.9760
Epoch 5/10
51/51 [==============================] - 11s 220ms/step - loss: 0.1907 - accuracy: 0.9529 - val_loss: 0.1210 - val_accuracy: 0.9760
Epoch 6/10
51/51 [==============================] - 10s 203ms/step - loss: 0.1907 - accuracy: 0.9529 - val_loss: 0.1186 - val_accuracy: 0.9760
Epoch 7/10
51/51 [==============================] - 10s 202ms/step - loss: 0.1905 - accuracy: 0.9529 - val_loss: 0.1241 - val_accuracy: 0.9760
Epoch 8/10
51/51 [==============================] - 12s 239ms/step - loss: 0.1905 - accuracy: 0.9529 - val_loss: 0.1172 - val_accuracy: 0.9760
Epoch 9/10
51/51 [==============================] - 12s 241ms/step - loss: 0.1907 - accuracy: 0.9529 - val_loss: 0.1281 - val_accuracy: 0.9760
Epoch 10/10
51/51 [==============================] - 10s 191ms/step - loss: 0.1902 - accuracy: 0.9529 - val_loss: 0.1236 - val_accuracy: 0.9760
```

```
from sklearn.metrics import classification_report

pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
109/109 [==============================] - 5s 45ms/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      3326
           1       0.00      0.00      0.00       138

    accuracy                           0.96      3464
   macro avg       0.48      0.50      0.49      3464
weighted avg       0.92      0.96      0.94      3464

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Unfortunately, the model did not ever correctly label the fraudulent listings as fraudulent, even when I resized the train and test data. Next we will try LSTM:

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 32)          320000

 lstm (LSTM)                 (None, 32)                8320

 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
51/51 [==============================] - 11s 211ms/step - loss: 0.1903 - accuracy: 0.9529 - val_loss: 0.1183 - val_accuracy: 0.9760
Epoch 2/10
51/51 [==============================] - 13s 262ms/step - loss: 0.1907 - accuracy: 0.9529 - val_loss: 0.1197 - val_accuracy: 0.9760
Epoch 3/10
51/51 [==============================] - 10s 203ms/step - loss: 0.1907 - accuracy: 0.9529 - val_loss: 0.1169 - val_accuracy: 0.9760
Epoch 4/10
51/51 [==============================] - 15s 293ms/step - loss: 0.1902 - accuracy: 0.9529 - val_loss: 0.1249 - val_accuracy: 0.9760
Epoch 5/10
51/51 [==============================] - 17s 343ms/step - loss: 0.1901 - accuracy: 0.9529 - val_loss: 0.1280 - val_accuracy: 0.9760
Epoch 6/10
51/51 [==============================] - 18s 346ms/step - loss: 0.1902 - accuracy: 0.9529 - val_loss: 0.1234 - val_accuracy: 0.9760
Epoch 7/10
51/51 [==============================] - 12s 237ms/step - loss: 0.1904 - accuracy: 0.9529 - val_loss: 0.1166 - val_accuracy: 0.9760
Epoch 8/10
51/51 [==============================] - 13s 254ms/step - loss: 0.1903 - accuracy: 0.9529 - val_loss: 0.1195 - val_accuracy: 0.9760
Epoch 9/10
51/51 [==============================] - 10s 195ms/step - loss: 0.1906 - accuracy: 0.9529 - val_loss: 0.1209 - val_accuracy: 0.9760
Epoch 10/10
51/51 [==============================] - 10s 195ms/step - loss: 0.1901 - accuracy: 0.9529 - val_loss: 0.1182 - val_accuracy: 0.9760
```

```
pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
109/109 [==============================] - 3s 30ms/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      3326
           1       0.00      0.00      0.00       138

    accuracy                           0.96      3464
   macro avg       0.48      0.50      0.49      3464
weighted avg       0.92      0.96      0.94      3464

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Again, no luck with the LSTM. Next lets look at the CNN.

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length=maxlen))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
```

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 500, 128)          1280000

 conv1d (Conv1D)             (None, 494, 32)           28704

 max_pooling1d (MaxPooling1D  (None, 98, 32)           0
 )

 conv1d_1 (Conv1D)           (None, 92, 32)            7200

 global_max_pooling1d (Globa  (None, 32)               0
 lMaxPooling1D)

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.le
```

```
history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
51/51 [==============================] - 33s 620ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 2/10
51/51 [==============================] - 38s 744ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 3/10
51/51 [==============================] - 39s 774ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 4/10
51/51 [==============================] - 33s 644ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 5/10
51/51 [==============================] - 35s 680ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 6/10
51/51 [==============================] - 37s 731ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 7/10
51/51 [==============================] - 36s 703ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 8/10
51/51 [==============================] - 30s 590ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 9/10
51/51 [==============================] - 29s 566ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
Epoch 10/10
51/51 [==============================] - 29s 570ms/step - loss: 0.7262 - accuracy: 0.9529 - val_loss: 0.3700 - val_accuracy: 0.9760
```

```
from sklearn.metrics import classification_report

pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
109/109 [==============================] - 3s 28ms/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      3326
           1       0.00      0.00      0.00       138

    accuracy                           0.96      3464
   macro avg       0.48      0.50      0.49      3464
weighted avg       0.92      0.96      0.94      3464

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Next we will try out some embeddings. Here we are adding an Embedding layer into the model.

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
```

```
history = model.fit(train_data, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 500, 8)            80000

 flatten (Flatten)           (None, 4000)              0

 dense_10 (Dense)            (None, 16)                64016

 dense_11 (Dense)            (None, 1)                 17

=================================================================
Total params: 144,033
Trainable params: 144,033
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
204/204 [==============================] - 2s 7ms/step - loss: 0.2001 - acc: 0.9529 - val_loss: 0.1133 - val_acc: 0.9760
Epoch 2/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1927 - acc: 0.9529 - val_loss: 0.1143 - val_acc: 0.9760
Epoch 3/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1922 - acc: 0.9529 - val_loss: 0.1157 - val_acc: 0.9760
Epoch 4/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1929 - acc: 0.9529 - val_loss: 0.1140 - val_acc: 0.9760
Epoch 5/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1922 - acc: 0.9529 - val_loss: 0.1466 - val_acc: 0.9760
Epoch 6/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1922 - acc: 0.9529 - val_loss: 0.1214 - val_acc: 0.9760
Epoch 7/10
204/204 [==============================] - 2s 8ms/step - loss: 0.1919 - acc: 0.9529 - val_loss: 0.1171 - val_acc: 0.9760
Epoch 8/10
204/204 [==============================] - 2s 10ms/step - loss: 0.1912 - acc: 0.9529 - val_loss: 0.1352 - val_acc: 0.9760
Epoch 9/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1922 - acc: 0.9529 - val_loss: 0.1550 - val_acc: 0.9760
Epoch 10/10
204/204 [==============================] - 1s 6ms/step - loss: 0.1926 - acc: 0.9529 - val_loss: 0.1153 - val_acc: 0.9760
```

In conclusion, for my dataset, it seems like the only working model was the sequential one. I think with a larger dataset, the results might have been better.

✓ 21s completed at 6:01 PM