In specific, A CNN is used. Convolutional Neural Networks (CNNs) are often considered better than Random Forests for specific types of tasks, particularly those involving high-dimensional data like images, videos, and spatial-temporal datasets.

CNNs automatically detect and learn the important features from raw data during the training process, thanks to their convolutional layers. This capacity for automatic feature learning is particularly effective for complex, high-dimensional data like images, where manually designing features is impractical.

Also, They are designed to scale well with high-dimensional data, making them suitable for tasks involving large images or videos. CNNs can manage this complexity without a significant increase in required computational resources relative to the size of the data.

Due to their architecture, CNNs are better at generalizing from the training data to new, unseen data, especially when it comes to images. This is because they learn abstract features that are more universally applicable across different visual domains. Random forest can be more likely to overfit to a dataset.

Different Setting is tested for this project. Epochs do not influence this model a lot due to it is rather a simple problem for the CNN model.

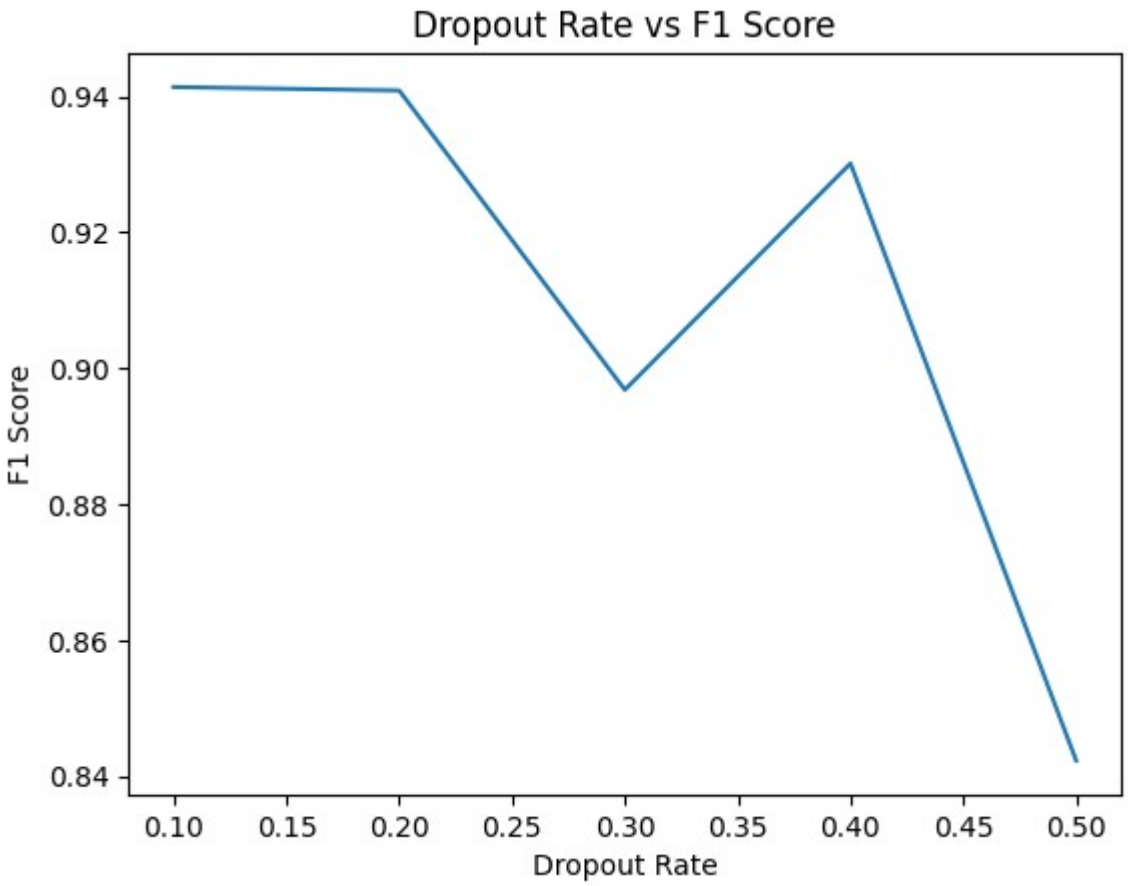
Dropout is a regularization technique used in the training of neural networks, including CNN, to prevent overfitting. The concept of dropout involves randomly "dropping out" a proportion of neurons (i.e., setting their activations to zero) in a layer during each training iteration. By "dropout rate," we refer to the probability that each neuron in the layer is dropped. For example, a dropout rate of 0.5 means there's a 50% chance that any given neuron will be set to zero during a particular forward or backward pass.

Dropout is not equally effective at all locations within a CNN. It's most commonly applied after fully connected layers or dense layers towards the end of the network. Applying dropout after convolutional layers is also possible, but it's typically done with a lower dropout rate, considering the different

nature of spatial feature extraction in these layers.

I Tested 5 different dropout rates, from 0.1 to 0.5.

```
Click to add a breakpoint.
#test different dropout rates
dropout_rates = [0.1, 0.2, 0.3, 0.4, 0.5]
f1_scores = []

for rate in dropout_rates:
    model = Sequential()
    model.add(Dense(64, input_dim=7, activation='relu'))
    model.add(Dropout(rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(rate))
    model.add(Dense(1, activation='sigmoid'))

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

    # fit the model
    history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_val, y_val))

    # use the model on validation data and evaluate
    y_pred = model.predict(X_val)
    y_pred = (y_pred > 0.5)

    # f1 score
    f1 = f1_score(y_val, y_pred)
    f1_scores.append(f1)

print(f1_scores)
```

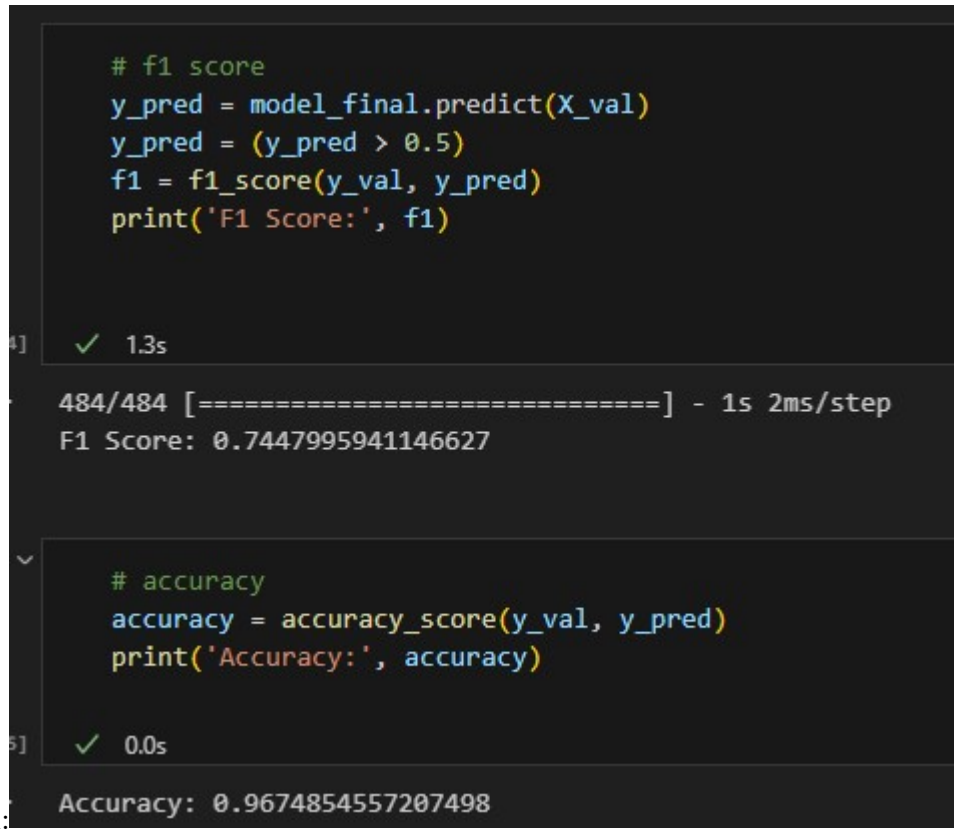


More Dropout rate seems to require higher epochs so 0.15 is chosen for this project. Here is the final look of the model itself:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_24 (Dense)            (None, 1024)              8192

 dropout_17 (Dropout)        (None, 1024)              0

 dense_25 (Dense)            (None, 512)               524800

 dropout_18 (Dropout)        (None, 512)               0

 dense_26 (Dense)            (None, 256)               131328

 dropout_19 (Dropout)        (None, 256)               0

 dense_27 (Dense)            (None, 128)               32896

 dropout_20 (Dropout)        (None, 128)               0

 dense_28 (Dense)            (None, 64)                8256

 dropout_21 (Dropout)        (None, 64)                0

 dense_29 (Dense)            (None, 32)                2080
...
Total params: 707585 (2.70 MB)
Trainable params: 707585 (2.70 MB)
Non-trainable params: 0 (0.00 Byte)
```

And here is the result of this

model:


# f1 score
y_pred = model_final.predict(X_val)
y_pred = (y_pred > 0.5)
f1 = f1_score(y_val, y_pred)
print('F1 Score:', f1)
1.3s
484/484 [==============================] - 1s 2ms/step
F1 Score: 0.7447995941146627
# accuracy
accuracy = accuracy_score(y_val, y_pred)
print('Accuracy:', accuracy)
0.0s
Accuracy: 0.9674854557207498