# **Models Trained with their respective parameters and metrics.**

|  |  |  |
| --- | --- | --- |
| Model Name | Parameters and Hyper Parameters | Metrics |
| CNN\_LSTM  (Court Trial Data)  100x64x64x1  Face detector:  Haarcascade\_frontalface  BIASED SPLIT | def build\_CNN\_LSTM(input\_shape):      model = Sequential()      # TimeDistributed wrapper to apply CNN across time dimension      model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input\_shape=input\_shape))      model.add(TimeDistributed(MaxPooling2D((2, 2))))      model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu')))      model.add(TimeDistributed(MaxPooling2D((2, 2))))      model.add(TimeDistributed(Flatten()))      # LSTM layer for temporal processing      model.add(LSTM(50, return\_sequences=False))      model.add(Dropout(0.3))      # Fully connected layers      model.add(Dense(100, activation='relu'))      model.add(Dropout(0.3))      model.add(Dense(1, activation='sigmoid'))        # Compile the model      model.compile(optimizer=Adam(),                loss='binary\_crossentropy',                metrics=['accuracy', Precision(), Recall(), AUC()])      return model | Accuracy: 91.6%  Loss: 0.34 |
| CNN\_LSTM  (MU3D Data)  100x64x64x1 | Same as above | Very poor  Accuracy: 30% ~ 50% |
| 3D CNN (Court Trial Data)  100x64x64x1  Face detector :  Haarcascade\_frontalface  BIASED SPLIT | model = Sequential()        # Convolutional layers      model.add(Conv3D(32, kernel\_size=(3, 3, 3), activation='relu'))      model.add(MaxPooling3D(pool\_size=(2, 2, 2)))        model.add(Conv3D(32, kernel\_size=(3, 3, 3), activation='relu'))      model.add(MaxPooling3D(pool\_size=(2, 2, 2)))      #model.add(Dropout(0.3))        model.add(Flatten())        # Fully connected layers      model.add(Dense(500, activation='relu'))      model.add(Dropout(0.3))      model.add(Dense(100, activation='relu'))      model.add(Dropout(0.3))        # Output layer      model.add(Dense(1, activation='sigmoid'))        # Compile the model      model.compile(optimizer=Adam(),                loss='binary\_crossentropy',                metrics=['accuracy', AUC()])        return model | Accuracy:  87.5%  Loss: 0.46 |
| 3D CNN (Court Trial Data)  100x64x64x1  NON-BIASED SPLIT  Face detector :  Haarcascade\_frontalface | Same as before | Accuracy:  83.33%  Loss: 0.46 |
| 3D CNN (Court Trial Data)  100x64x64x1  NON-BIASED SPLIT  Face detector:  MTCNN  [BEST MODEL] | def build\_3D\_CNN\_ALT():      model = Sequential()        # Convolutional layers      model.add(Conv3D(32, kernel\_size=(1, 3, 3), activation='relu'))      model.add(MaxPooling3D(pool\_size=(2, 2, 2)))        model.add(Conv3D(32, kernel\_size=(3, 1, 1), activation='relu'))      model.add(MaxPooling3D(pool\_size=(2, 2, 2)))          model.add(Flatten())        # Fully connected layers      model.add(Dense(500, activation='relu'))      model.add(Dropout(0.3))      model.add(Dense(100, activation='relu'))      model.add(Dropout(0.3))        # Output layer      model.add(Dense(1, activation='sigmoid'))        #Compile the model      model.compile(optimizer=RMSprop(),                loss='binary\_crossentropy',                metrics=['accuracy', AUC()])      return model | Accuracy:  87.5%  Loss: 0.61 |
| CNN-LSTM (Court Trial Data)  100x64x64x1  NON-BIASED SPLIT  Face detector :  MTCNN | # CNN-LSTM alternative architecture (works will with MTCNN)  def build\_CNN\_LSTM\_ALT(input\_shape):      model = Sequential()      # TimeDistributed wrapper to apply CNN across time dimension      model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu'), input\_shape=input\_shape))      model.add(TimeDistributed(MaxPooling2D((2, 2))))      model.add(Dropout(0.3))      model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu')))      model.add(TimeDistributed(MaxPooling2D((2, 2))))      model.add(TimeDistributed(Flatten()))      model.add(Dropout(0.3))      # LSTM layer for temporal processing      model.add(LSTM(100, return\_sequences=False))      model.add(Dropout(0.3))        # Fully connected layers      model.add(Dense(500, activation='relu'))      model.add(Dropout(0.5))      model.add(Dense(100, activation='relu'))      model.add(Dropout(0.3))      model.add(Dense(1, activation='sigmoid'))        #Compile the model      model.compile(optimizer=RMSprop(),                loss='binary\_crossentropy',                metrics=['accuracy', AUC()])        return model | Accuracy:  79.17%  Loss: 0.65 |

Very important remark: the results with MTCNN face extractor are to be chosen for our proposed method because MTCNN is consistent with extracting the faces accurately unlike Haar Cascades Frontal Face detector which sometimes fails and leads to unpredictable results (or even different outputs for the same input)

Note, on real life trial, using other sampling methods like splitting video into smaller videos with different labels gives worse results, also when sampling the whole videos into more than 100 frames or with a bigger frame size, results also seem to be worse.

MU3D didn’t achieve any usable results regardless of the preprocessing combinations and models architectures that were tried.