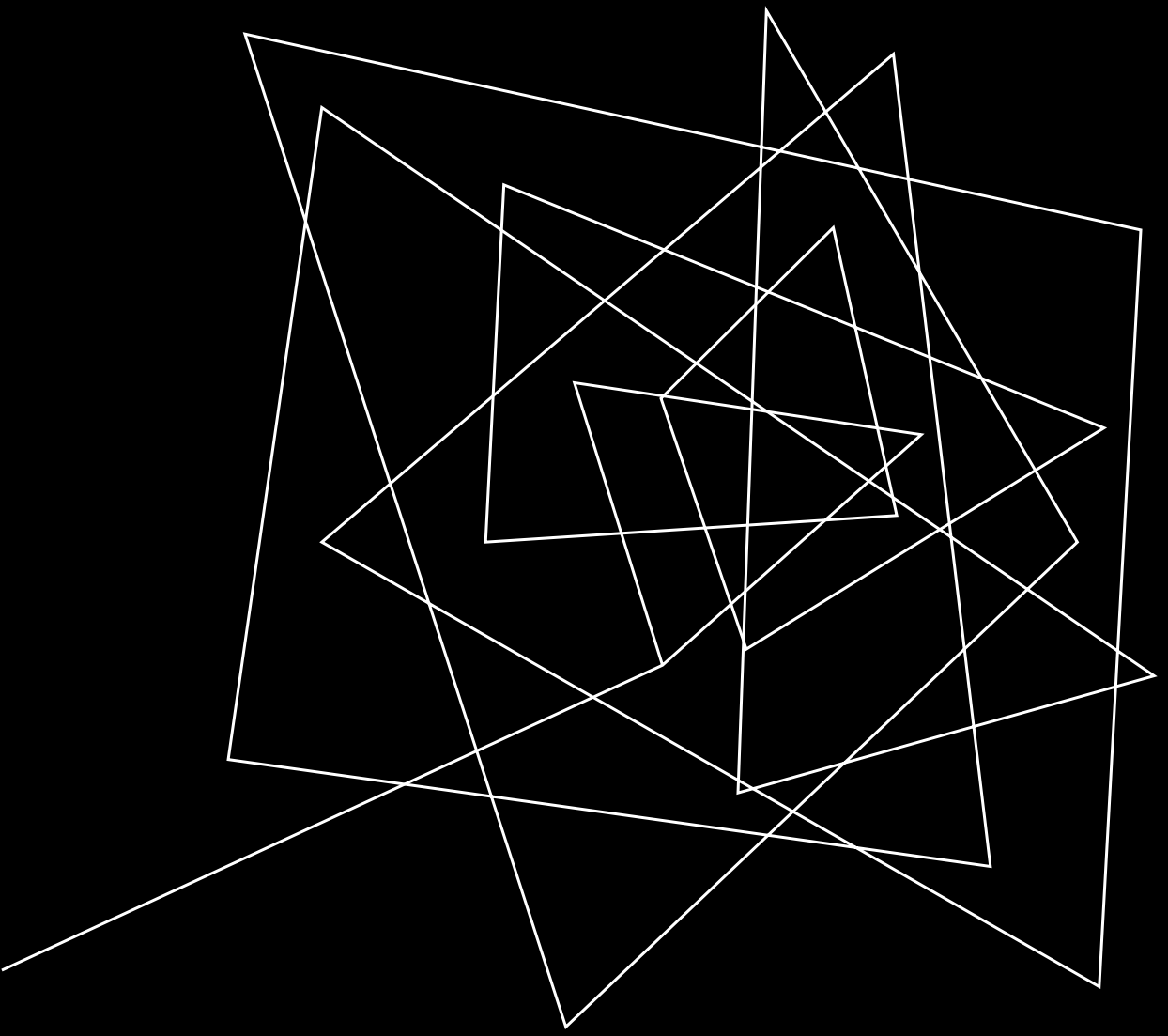


PREDICTING ICU MORTALITY USING DEEP LEARNING (FCNN)

MAZIYAR MIRZAEI
FALL 2024
UT-AUSTIN

INTRODUCTION

Objective: Predict ICU mortality using patient data

Approach: Fully Connected Neural Network (FCNN)

Dataset: MIMIC-III (Medical Information Mart for Intensive Care)

A step-by-step guide on building a Fully Connected Neural Network for mortality prediction in ICU patients.

DATASET OVERVIEW (MIMIC-III)

- Patient Information: Gender, Age, Expire Flag, etc.
- Admission Details: Admit time, Discharge time, Diagnosis, etc.
- ICU Stay: ICU entry/exit times, Length of Stay (LOS)
- Vital Signs & Lab Results: Heart Rate, Blood Pressure, Sodium, Potassium, etc.

PROJECT OVERVIEW

- **Goal:** To predict ICU mortality using patient data (vital signs and lab results).
- **Data Source:** ICU data including patient demographics, vitals, and lab tests.
- **Steps:**
  - Data Collection & Preprocessing
  - Model Design (FCNN)
  - Training the Model
  - Evaluating Performance

DATA PREPROCESSING

- Merging patient, admission, and ICU stay data
- Adding the MORTALITY flag as the target variable
- Filtering data for the first 24 hours of ICU stay

```
import pandas as pd
# Load data
patients = pd.read_csv('PATIENTS.csv', low_memory=False)
admissions = pd.read_csv('ADMISSIONS.csv',
low_memory=False)
icustays = pd.read_csv('ICUSTAYS.csv', low_memory=False)
chartevents = pd.read_csv('CHARTEVENTS.csv',
low_memory=False)
labevents = pd.read_csv('LABEVENTS.csv',
low_memory=False)

# Merge patient, admission, and ICU stay data
merged_df = icustays.merge(admissions, on=['SUBJECT_ID',
'HADM_ID'], how='inner') \
                .merge(patients, on='SUBJECT_ID',
how='inner')

# Add mortality flag as the target variable
merged_df['MORTALITY'] =
merged_df['HOSPITAL_EXPIRE_FLAG']
```

FEATURE EXTRACTION

Feature Selection (Vital Signs & Labs)

Slide Content:

• Vital signs: Heart Rate, Systolic BP, Diastolic BP, Respiratory Rate, Oxygen Saturation

• Lab tests: Sodium, Potassium, Bicarbonate, etc.

• Aggregated over the first 24 hours: Mean, Max, Min

```
# Aggregating vital sign data (mean, min, max)
vital_signs_itemids = [220045, 220179, 220180,
220181, 220210, 220277]
vital_agg = filtered_vitals.groupby(['ICUSTAY_ID',
'ITEMID']).agg({
    'VALUENUM': ['mean', 'max', 'min']
}).reset_index()

# Pivot to create one row per ICU stay
vital_features =
vital_agg.pivot(index='ICUSTAY_ID',
columns='ITEMID', values=['mean', 'max', 'min'])
vital_features.columns = ['_'.join(map(str, col))
for col in vital_features.columns]
merged_df = pd.merge(merged_df, vital_features,
on='ICUSTAY_ID', how='left')
```

DATA SPLITTING AND SCALING

- Apply `StandardScaler` to standardize features
- Ensures that all features have mean 0 and variance 1

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Define features and target
X_fcnn = merged_df[vital_cols + lab_cols].values
y_fcnn = merged_df['MORTALITY'].values

# Standardize the features
scaler = StandardScaler()
X_fcnn = scaler.fit_transform(X_fcnn)

# Split into training and test sets
X_train_fcnn, X_test_fcnn, y_train_fcnn, y_test_fcnn =
train_test_split(X_fcnn, y_fcnn, test_size=0.3,
random_state=42)
```

EVALUATING THE PERFORMANCE OF THE DEEP LEARNING MODEL

```
from sklearn.metrics import accuracy_score, roc_auc_score,
classification_report

# Make predictions
y_pred = (model.predict(X_test_scaled) >
0.5).astype("int32")

# Calculate accuracy and AUC-ROC score
accuracy = accuracy_score(y_test, y_pred)
y_pred_proba = model.predict(X_test_scaled)

# Probabilities for AUC
roc_auc = roc_auc_score(y_test, y_pred_proba)

print(f"Deep Learning Model Accuracy: {accuracy}")
print(f"Deep Learning Model AUC-ROC: {roc_auc}")

# Classification report
print(classification_report(y_test, y_pred))
```

```
577/577 ━━━━━━━━━━━━━━━━━━━━ 0s 223us/step
577/577 ━━━━━━━━━━━━━━━━━━━━ 0s 187us/step
Deep Learning Model Accuracy: 0.8957204767063922
Deep Learning Model AUC-ROC: 0.7504477567662016
              precision    recall  f1-score   support

           0       0.90      1.00      0.94     16494
           1       0.75      0.03      0.06      1966

    accuracy                           0.90     18460
   macro avg       0.82      0.52      0.50     18460
weighted avg       0.88      0.90      0.85     18460
```

FCNN MODEL DESIGN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assume merged_df, vital_cols, and lab_cols are defined and X_lstm and y_lstm are
prepared
# Combine the features for the FCNN
X_fcnn = merged_df[vital_cols + lab_cols].values  # All features (vitals and labs)
y_fcnn = merged_df['MORTALITY'].values            # Target variable

# Standardize the features
scaler = StandardScaler()
X_fcnn = scaler.fit_transform(X_fcnn)

# Split the data into training and testing sets
X_train_fcnn, X_test_fcnn, y_train_fcnn, y_test_fcnn = train_test_split(X_fcnn,
y_fcnn, test_size=0.3, random_state=42)

# Define a Fully Connected Neural Network (FCNN) model
fcnn_model = Sequential()

# First dense layer with 128 units
fcnn_model.add(Dense(128, activation='relu', input_shape=(X_train_fcnn.shape[1],)))
fcnn_model.add(Dropout(0.3))  # Dropout layer to prevent overfitting

# Second dense layer with 64 units
fcnn_model.add(Dense(64, activation='relu'))
fcnn_model.add(Dropout(0.3))

# Third dense layer with 32 units
fcnn_model.add(Dense(32, activation='relu'))
fcnn_model.add(Dropout(0.3))

# Output layer with sigmoid activation for binary classification
fcnn_model.add(Dense(1, activation='sigmoid'))

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

# Print model summary
fcnn_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 128)	3,584
dropout_8 (Dropout)	(None, 128)	0
dense_12 (Dense)	(None, 64)	8,256
dropout_9 (Dropout)	(None, 64)	0
dense_13 (Dense)	(None, 32)	2,080
dropout_10 (Dropout)	(None, 32)	0
dense_14 (Dense)	(None, 1)	33

Total params: 13,953 (54.50 KB)

Trainable params: 13,953 (54.50 KB)

Non-trainable params: 0 (0.00 B)

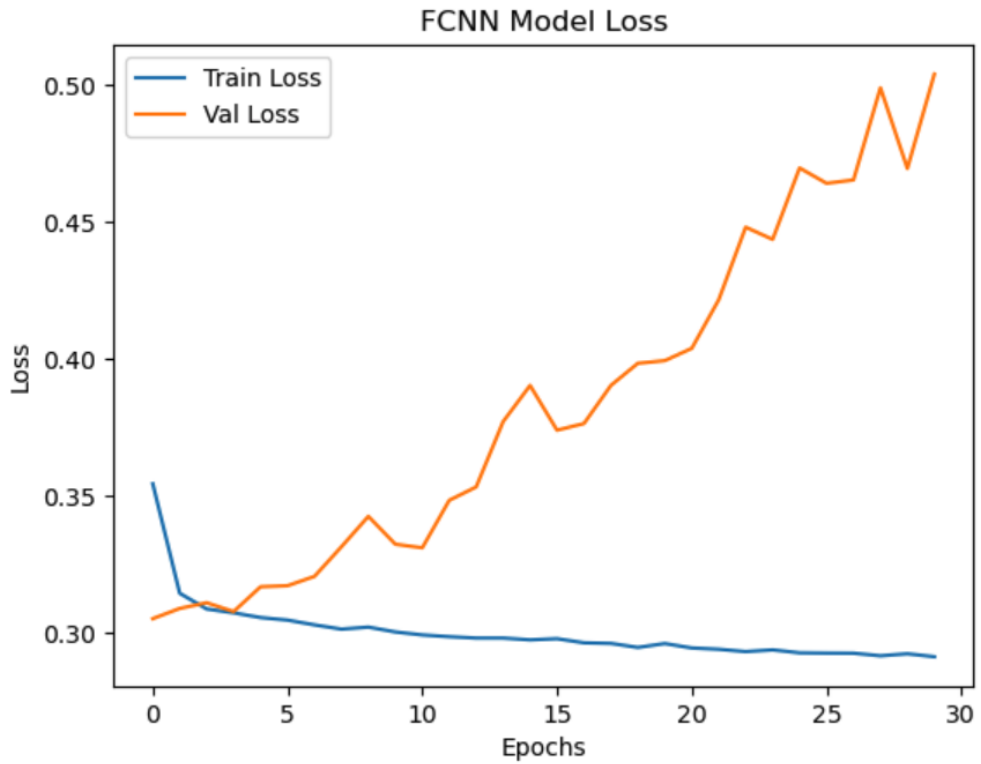
MODEL TRAINING

```
# Train the model
history_fcnn = fcnn_model.fit(X_train_fcnn, y_train_fcnn, epochs=30, batch_size=64,
validation_data=(X_test_fcnn, y_test_fcnn))
```

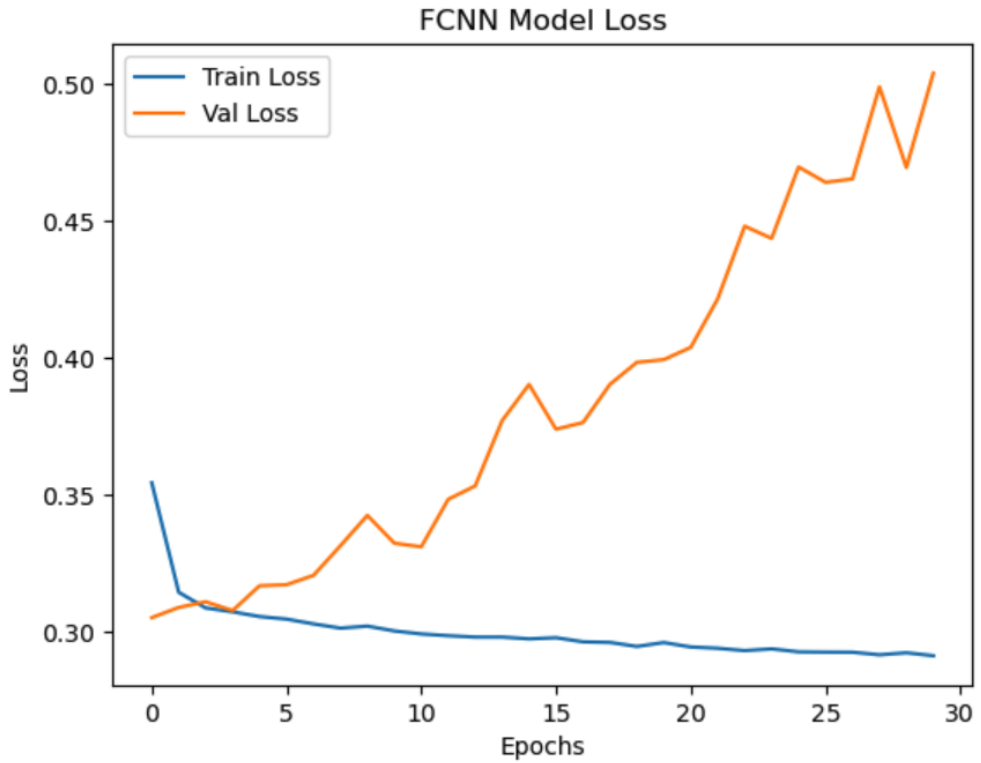
```
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 483us/step - accuracy: 0.8921 - loss: 0.3016 - val_accuracy: 0.8959 - val_loss: 0.4216
Epoch 23/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 486us/step - accuracy: 0.8955 - loss: 0.2934 - val_accuracy: 0.8964 - val_loss: 0.4481
Epoch 24/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 545us/step - accuracy: 0.8957 - loss: 0.2949 - val_accuracy: 0.8960 - val_loss: 0.4437
Epoch 25/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 483us/step - accuracy: 0.8947 - loss: 0.2961 - val_accuracy: 0.8962 - val_loss: 0.4698
Epoch 26/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 477us/step - accuracy: 0.8947 - loss: 0.2958 - val_accuracy: 0.8963 - val_loss: 0.4642
Epoch 27/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 485us/step - accuracy: 0.8947 - loss: 0.2943 - val_accuracy: 0.8966 - val_loss: 0.4654
Epoch 28/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 479us/step - accuracy: 0.8996 - loss: 0.2859 - val_accuracy: 0.8967 - val_loss: 0.4991
Epoch 29/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 482us/step - accuracy: 0.8964 - loss: 0.2923 - val_accuracy: 0.8969 - val_loss: 0.4696
Epoch 30/30
673/673 ━━━━━━━━━━━━━━━━━━━━ 0s 561us/step - accuracy: 0.8963 - loss: 0.2887 - val_accuracy: 0.8966 - val_loss: 0.5041
```

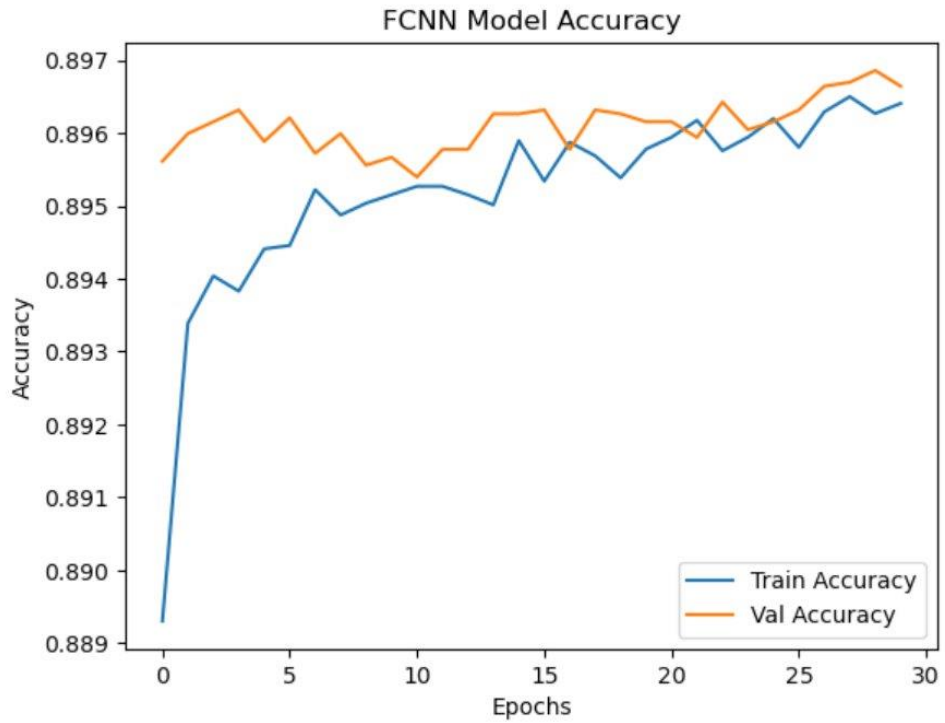
VISUALIZING MODEL ACCURACY AND LOSS

```
# Plot the training history (accuracy)
plt.plot(history_fcnn.history['accuracy'], label='Train
Accuracy')
plt.plot(history_fcnn.history['val_accuracy'], label='Val
Accuracy')
plt.title('FCNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training history (loss)
plt.plot(history_fcnn.history['loss'], label='Train
Loss')
plt.plot(history_fcnn.history['val_loss'], label='Val
Loss')
plt.title('FCNN Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate on the test set
test_loss, test_accuracy =
fcnn_model.evaluate(X_test_fcnn, y_test_fcnn)
print(f"Test Accuracy: {test_accuracy}")
```




```
577/577 ━━━━━━━━━━━━━━━━━━━━ 0s 210us/step - accuracy: 0.8990 - loss: 0.5653
Test Accuracy: 0.8966413736343384
```



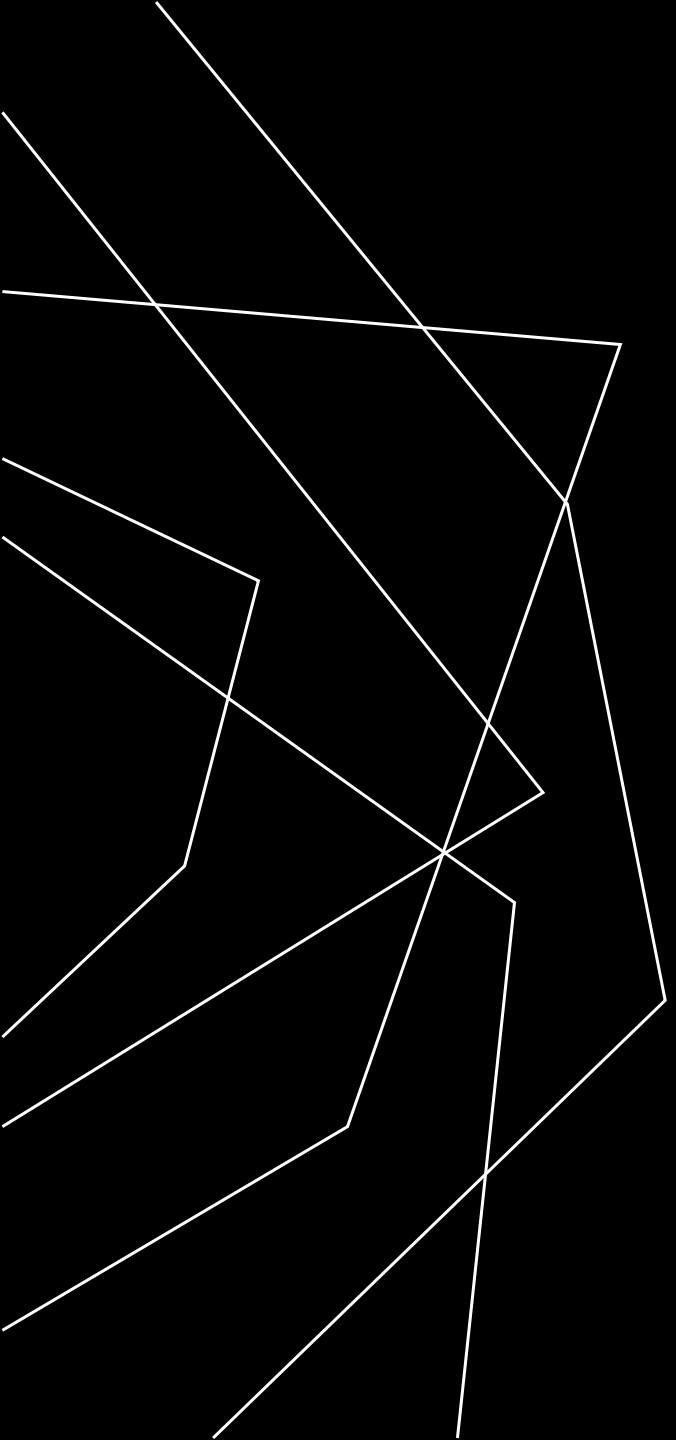



```
577/577 ━━━━━━━━━━━━━━━━━━━━ 0s 210us/step - accuracy: 0.8990 - loss: 0.5653
Test Accuracy: 0.8966413736343384
```

CONCLUSION

Test Accuracy: ~0.897

- **Key Points:**
  - FCNN achieved high accuracy (~89%) in predicting ICU mortality.
  - Model generalizes well but shows some overfitting signs.
  - Further improvements can be made by tuning hyperparameters or using different models.



THANK YOU

Maziyar Mirzaei