Radar Pulse Detection using Fourier Transform and Convolutional Neural Network (FT-CNN)

Group 3 Members:

Aparna Bharani - CB.SC.U4AIE24304

I Mahalakshmi - CB.SC.U4AIE24322

Maalika P - CB.SC.U4AIE24332

Parkavi R - CB.SC.U4AIE24338

Abstract:

This project explores radar pulse detection by applying the Fourier Transform (FT) to radar signals and analyzing the resulting frequency components using three neural network models: FT-CNN (Convolutional Neural Network), FT-PNN (Probabilistic Neural Network), and FT-BPNN (Backpropagation Neural Network). We used the Ionosphere Dataset from the UCI repository to train and test these models. The results show that FT-CNN delivers the highest accuracy, highlighting its superior capability

Introduction:

Radar pulse signal detection plays a crucial role in modern electronic warfare, spectrum monitoring, and aerospace communication systems. Traditional signal detection techniques often rely on hand-crafted features and threshold-based decision methods, which may falter under low signal-to-noise ratio (SNR) or complex signal environments

Objective:

To design and implement a **radar pulse signal detection system** by integrating **Fourier Transform (FT)** with a **Convolutional Neural Network (CNN)**.

- The **Fast Fourier Transform (FFT)** converts raw time-domain radar signal data into the frequency domain.
- These transformed signals are then used to train a deep CNN model that **automatically learns frequency-based patterns** for accurate classification of radar pulse presence.
- The model is evaluated using **precision**, **recall**, and **F1-score** to ensure **high detection reliability**, even in noisy or uncertain environments.

Data Description:

- Dataset: **Ionosphere Dataset** (from the UCI Machine Learning Repository)
- Instances: **351**
- Features: **34**
- Description: Radar signal measurements.
- Target Labels:
- `'g'` → Good signal
- `'b'` → Bad signal -Data Characteristics: -Real-Valued Data -No Missing Values -Imbalanced Dataset

Model: FT - CNN

Combining **Fourier-transformed features** with a **deep CNN** to detect the presence of radar pulses effectively.

Methodology:

Data Preprocessing

-Split the dataset → 80% for training, 20% for testing

-Normalized data → To ensure all feature values are in the same range -Missing values → Filled with mean values

Feature Extraction — Fourier Transform (FT)

-FT converts time domain to frequency domain

-Better capture of signal patterns and periodic features

-Used absolute value of FT output. CNN Flow

1.Data Preprocessing:

-FFT applied on Ionosphere Dataset

-Data Normalization

-Reshape into 1xNx1 format suitable for CNN

2.Feature Extraction:

-Convolution Layers detect local patterns

-Swish Activation improves gradient flow

3.Dimensionality Reduction:

-MaxPoolingreduces feature size

-Controls overfitting

4.Classification:

-Fully Connected Layers learn global patterns

-SoftmaxLayer provides final class probabili

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Dense, Flatten, Dropout, MaxPooling1D
from scipy.fft import fft
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error
import seaborn as sns
from collections import Counter
```

1.Load and Prepare Dataset

```
dataset_path = r"C:\Users\parka\OneDrive\Documents\mfc_eoc_proj\mfc_eoc\ionosphere.csv"
dataset = pd.read_csv(dataset_path)
```

2.Extract features (X) and labels (y)

```
X = dataset.iloc[:, :-1].values # Radar signal data
y = dataset.iloc[:, -1].map({'g': 1, 'b': 0}).values # Convert labels to binary
```

3.Apply Fourier Transform (FFT)

```
X_ft = np.abs(fft(X, axis=1)) # Convert to frequency domain
```

4.Normalize Data

```
scaler = StandardScaler()

X_ft = scaler.fit_transform(X_ft)
```

5.Train a CNN Model

```
X_train, X_test, y_train, y_test = train_test_split(X_ft, y, test_size=0.2, random_state=42)

print(len(X_test)) # Check how many test samples

print(y_test) # Class distribution

print(Counter(y_test))
```

6.Reshape data for CNN input

```
X_train = X_train[..., np.newaxis]

X_test = X_test[..., np.newaxis]
```

7.Build CNN Model using Swish activation

```
model = Sequential([ Conv1D(32, kernel_size=3, activation='swish', input_shape=(X_train.shape[1], 1)),

MaxPooling1D(2),

Conv1D(64, kernel_size=3, activation='swish'),

MaxPooling1D(2),

Flatten(),

Dense(128, activation='swish'),

Dropout(0.5),

Dense(1, activation='sigmoid') ])
```

8.Compile Model

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

9.Train Model

model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

10.Evaluate Model

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy:.4f}")

11.Visualize Results

plt.figure(figsize=(10, 4))

plt.imshow(X_ft, aspect='auto', cmap='hot')

plt.colorbar()

plt.title("Fourier Transformed Radar Signal Data")

plt.xlabel("Features (Frequency Components)")

plt.ylabel("Samples")

plt.show()

12.Predictions

y_pred_prob = model.predict(X_test)

y_pred = (y_pred_prob > 0.5).astype(int)

13.Compute Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

Plot Confusion Matrix

plt.figure(figsize=(6,5))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['b (Bad)', 'g (Good)'], yticklabels=['b (Bad)', 'g (Good)'])

```
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix for FT-CNN with Swish Activation')

plt.show()
```

14.Classification Report

```
print("\nClassification Report:")

print(classification_report(y_test, y_pred))
```

15.Mean Squared Error (MSE)

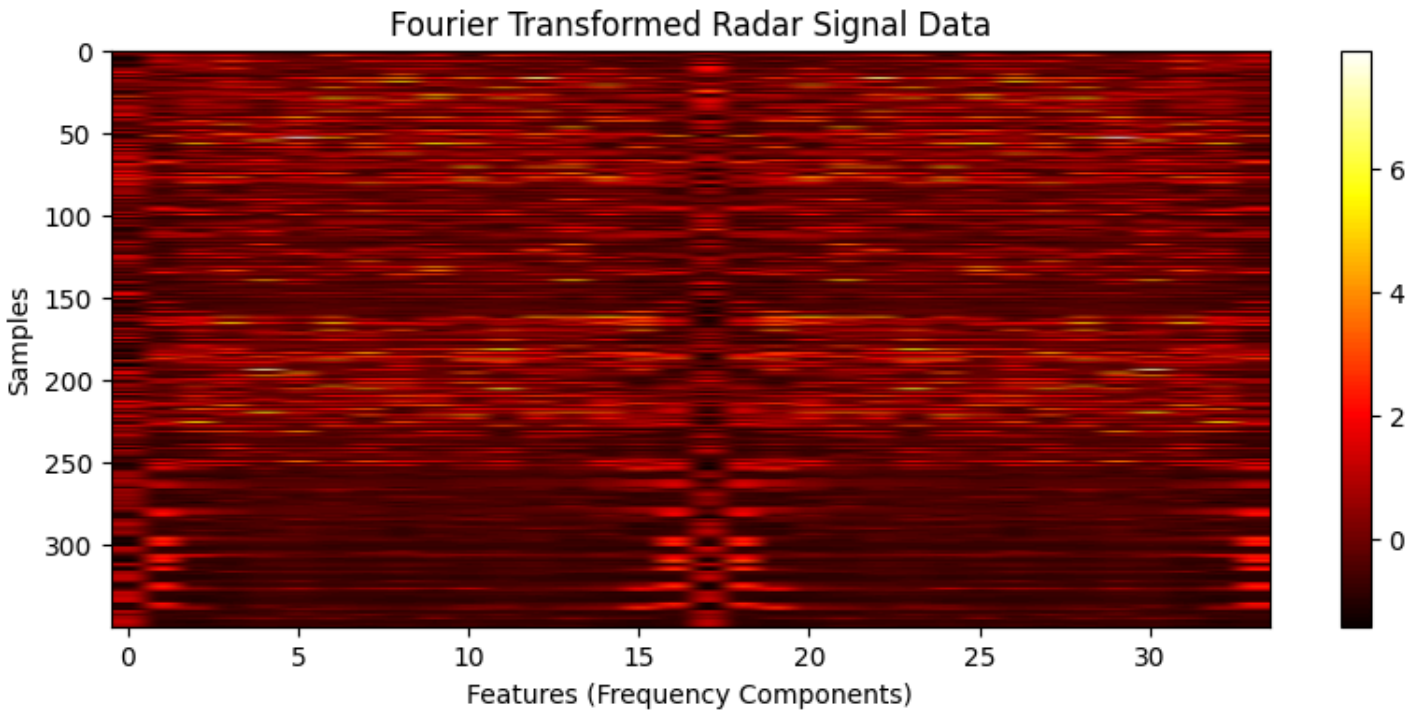
```
mse = mean_squared_error(y_test, y_pred_prob)

print(f"\nMean Squared Error (MSE): {mse:.6f}")
```

16.Plot a radar signal sample

```
plt.figure(figsize=(10, 4))

plt.plot(X_test[0], label="Radar Signal Sample")

plt.title(f"Predicted: {y_pred[0][0]}, Actual: {y_test[0]}")

plt.legend()

plt.show()
```

```
70
[0 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 0 0 0 1 1 0
 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1]
Counter({np.int64(1): 44, np.int64(0): 26})
C:\Users\parka\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\lo
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/60

1/9 ━                   1:48 14s/step - accuracy: 0.4688 - loss: 0.8789

3/9 ━━━━                0s 30ms/step - accuracy: 0.6302 - loss: 0.6461

5/9 ━━━━━━━             0s 30ms/step - accuracy: 0.7041 - loss: 0.5312
```

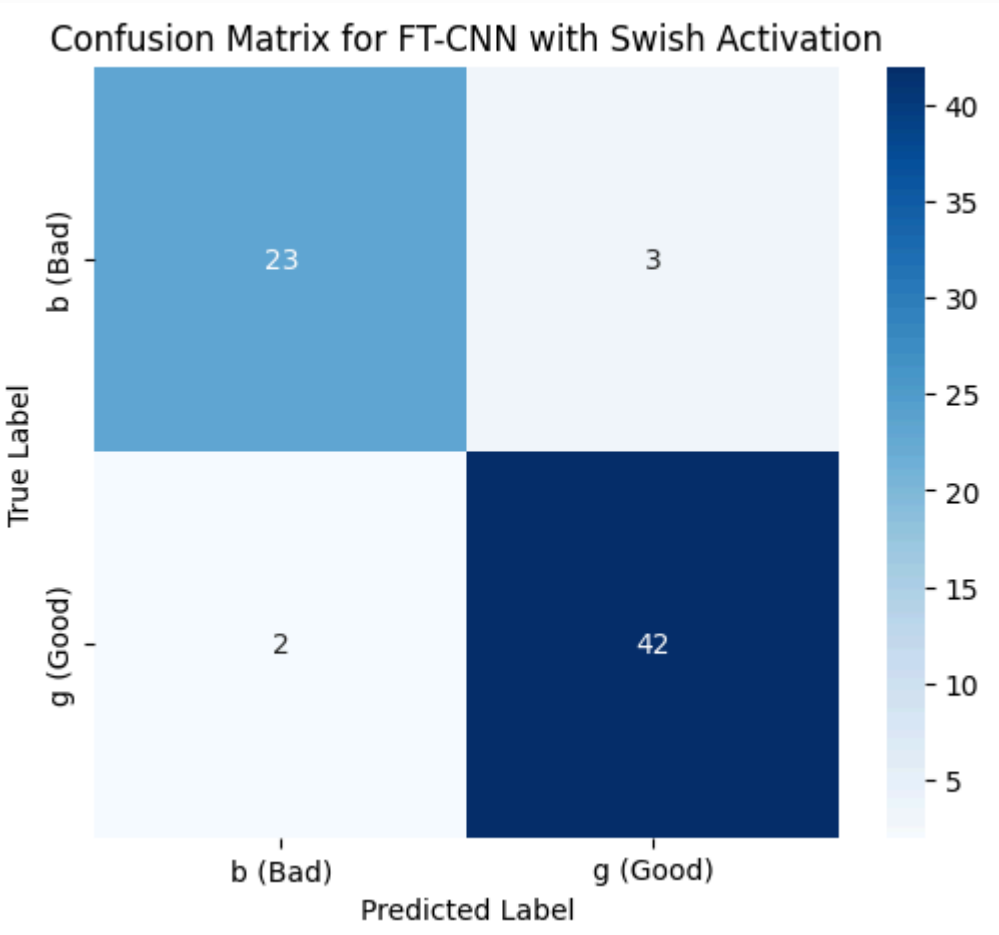
```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 1.0000 - loss: 4.2964e-05

9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 1.0000 - loss: 4.4196e-05 - val_accuracy: 0.8857 - v
Epoch 57/60

1/9 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - accuracy: 1.0000 - loss: 8.5147e-05

3/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 1.0000 - loss: 9.2702e-05

6/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 1.0000 - loss: 1.0111e-04

9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 1.0000 - loss: 1.0179e-04

9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 1.0000 - loss: 1.0182e-04 - val_accuracy: 0.9000 - v
Epoch 58/60

1/9 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step - accuracy: 1.0000 - loss: 3.3638e-05

3/9 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 1.0000 - loss: 3.1343e-05

6/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 1.0000 - loss: 2.7254e-05

9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 1.0000 - loss: 3.7543e-05

9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 1.0000 - loss: 4.0785e-05 - val_accuracy: 0.9000 - v
Epoch 59/60

1/9 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step - accuracy: 1.0000 - loss: 1.4324e-04

3/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 1.0000 - loss: 1.7830e-04

6/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 1.0000 - loss: 2.2887e-04

9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 1.0000 - loss: 2.1593e-04

9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 1.0000 - loss: 2.1176e-04 - val_accuracy: 0.9286 - v
Epoch 60/60

1/9 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 1.0000 - loss: 2.1209e-04

3/9 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 1.0000 - loss: 1.4749e-04

5/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 1.0000 - loss: 1.3806e-04

8/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 1.0000 - loss: 1.3192e-04

9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 1.3719e-04 - val_accuracy: 0.9286 - v

1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.9375 - loss: 0.4556

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.9330 - loss: 0.3866
Test Accuracy: 0.9286
```



1/3 ━━━━ **1s** 632ms/step

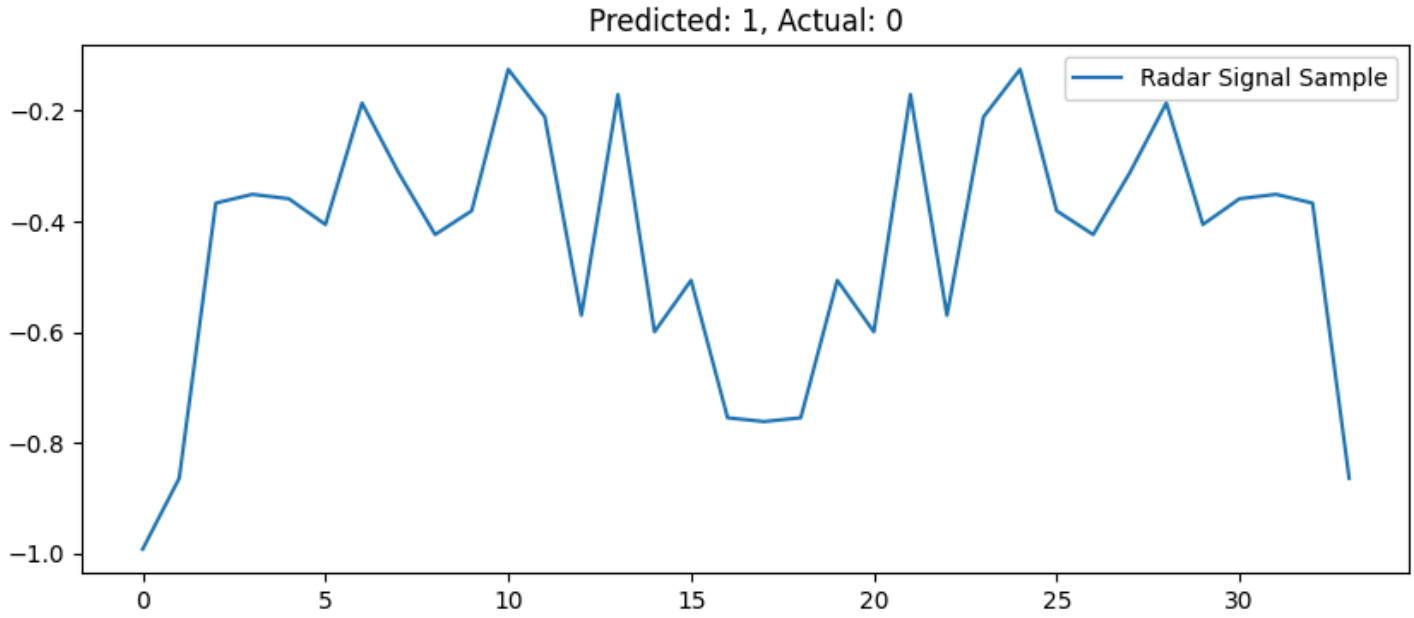
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 279ms/step

3/3 ━━━━━━━━━━━━━━━━━━━━ **1s** 316ms/step




```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.88      0.90        26
           1       0.93      0.95
```

```
              precision    recall  f1-score   support

    accuracy                         0.93        70
   macro avg       0.93      0.92      0.92        70
weighted avg       0.93      0.93      0.93        70


Mean Squared Error (MSE): 0.066759
```



Model Comparison

This FT-CNN model is compared with two other models:

- **FT-PNN**: Fourier Transform + Probabilistic Neural Network
- **FT-BPNN**: Fourier Transform + Back-propagation Neural Network

---

FT-PNN (Fourier Transform + Probabilistic Neural Network)

- The FT-PNN model applies a **Probabilistic Neural Network (PNN)** to the Fourier-transformed radar signal data.
- It uses **Gaussian kernels** to estimate the probability density function (PDF) of each class.
- Classification is based on **Bayesian decision theory**.
- **Non-iterative**, making it **fast to train**.
- Highly effective for **small to medium datasets** with high-dimensional features (like frequency domain data).

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.fft import fft

from scipy.stats import multivariate_normal

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc, mean_squared_error
```

1.Load Dataset

df = pd.read_csv(r"C:\Users\parka\OneDrive\Documents\mfc_eoc_proj\mfc_eoc\ionosphere.csv") # Use your actual dataset file

2.Extract Features & Labels

feature_columns = df.columns[:-1] # All but last column as features

label_column = df.columns[-1] # Last column as label

X = df[feature_columns].values

y = df[label_column].values

3.Compute Fourier Transform Features

FT_features = np.abs(fft(X, axis=1)) # Compute magnitude spectrum

FT_features = FT_features[:, :FT_features.shape[1] // 2] # Keep only first half (FFT symmetry)

4.Normalize Data

scaler = StandardScaler()

FT_features = scaler.fit_transform(FT_features)

5.Split Data into Train & Test

X_train, X_test, y_train, y_test = train_test_split(FT_features, y, test_size=0.2, random_state=42)

6.Implement Probabilistic Neural Network (PNN)

def pnn_predict(X_train, y_train, X_test, sigma=0.5):

classes = np.unique(y_train)

predictions = []

for x in X_test:

```
probabilities = []

for c in classes:

X_class = X_train[y_train == c] # Get samples of class c

prob = np.mean([multivariate_normal.pdf(x, mean=xi, cov=sigma**2) for xi in X_class])

probabilities.append(prob)

predictions.append(classes[np.argmax(probabilities)]) # Class with highest probability

return np.array(predictions)
```

7.Train & Predict with PNN

```
y_pred = pnn_predict(X_train, y_train, X_test, sigma=0.5)
```

8.Evaluate Model Performance

9.Accuracy

```
accuracy = accuracy_score(y_test, y_pred)

print(f'FT-PNN Accuracy: {accuracy * 100:.2f}%')
```

10.Mean Squared Error (MSE)

```
y_test_num = (y_test == np.unique(y_test)[1]).astype(int) # Convert to numeric (0/1)

y_pred_num = (y_pred == np.unique(y_test)[1]).astype(int)

mse = mean_squared_error(y_test_num, y_pred_num)

print(f'Mean Squared Error (MSE): {mse:.4f}')
```

11.Confusion Matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
yticklabels=np.unique(y_test))

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('FT-PNN Confusion Matrix')

plt.show()
```

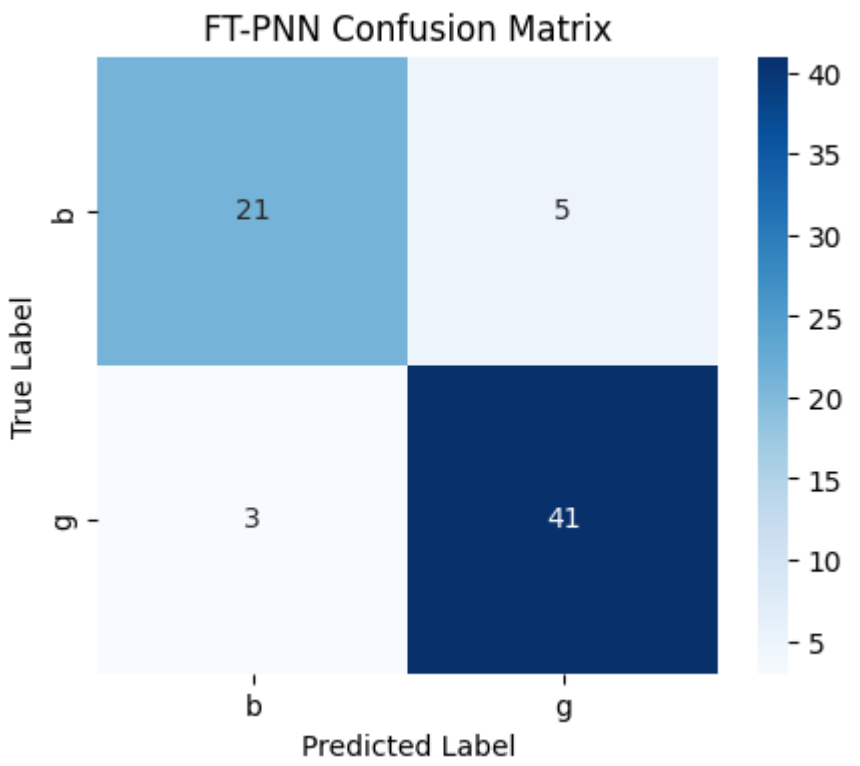
12.ROC Curve

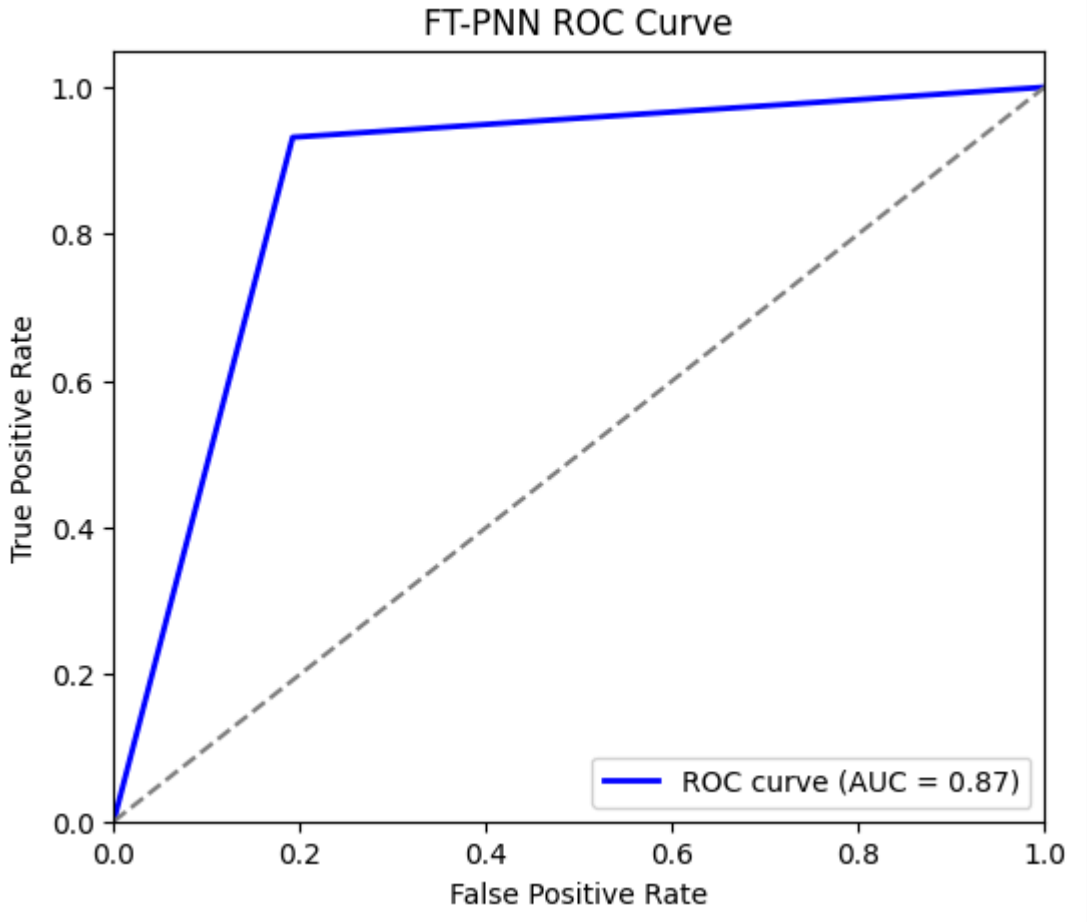
```
fpr, tpr, _ = roc_curve(y_test_num, y_pred_num)

roc_auc = auc(fpr, tpr)
```

13.Plot ROC Curve

```
plt.figure(figsize=(6,5))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('FT-PNN ROC Curve')

plt.legend(loc="lower right")

plt.show()
```

```
🔹 FT-PNN Accuracy: 88.57%
🔹 Mean Squared Error (MSE): 0.1143
```





FT-BPNN (Fourier Transform + Back-Propagation Neural Network)

- The FT-BPNN model applies a **Feedforward Neural Network (FNN)** trained using the **backpropagation algorithm** to the Fourier-transformed radar signal data.
- It **learns complex, non-linear patterns** in the frequency domain by **iteratively adjusting weights** to minimize classification error.
- The model uses a **loss function** (like cross-entropy or MSE) to guide training.
- Requires **sufficient training data** and **computational resources**, but is highly flexible and powerful.
- Suitable for tasks involving complex, high-dimensional frequency features.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import roc_curve, auc, confusion_matrix, ConfusionMatrixDisplay

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical
```

1. Load dataset

```
data = pd.read_csv(r"C:\Users\parka\OneDrive\Documents\mfc_eoc_proj\mfc_eoc\ionosphere.csv")
```

2. Split features and labels

```
X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values
```

3. Apply Fourier Transform (magnitude only)

```
X_ft = np.abs(np.fft.fft(X, axis=1))
```

4. Normalize the features

```
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_ft)
```

5. Encode labels

```
le = LabelEncoder()

y_encoded = le.fit_transform(y)

y_categorical = to_categorical(y_encoded)
```

6. Train-test split

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_categorical, test_size=0.2, random_state=42)
```

7. Build BPNN model

```
model = Sequential()

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(y_categorical.shape[1], activation='softmax'))
```

8. Compile the model

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

9. Train the model

```
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))
```

10. Evaluate model performance

```
loss, accuracy = model.evaluate(X_test, y_test)

print(f"\n Test Accuracy: {accuracy:.4f}")
```

11. Prediction and label decoding

```
y_score = model.predict(X_test)

y_pred = np.argmax(y_score, axis=1)

y_true = np.argmax(y_test, axis=1)
```

12. ROC Curve (macro-average)

```
fpr, tpr, _ = roc_curve(y_test.ravel(), y_score.ravel())

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.title('FT-BPNN ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc='lower right')

plt.grid(True)

plt.tight_layout()

plt.show()
```

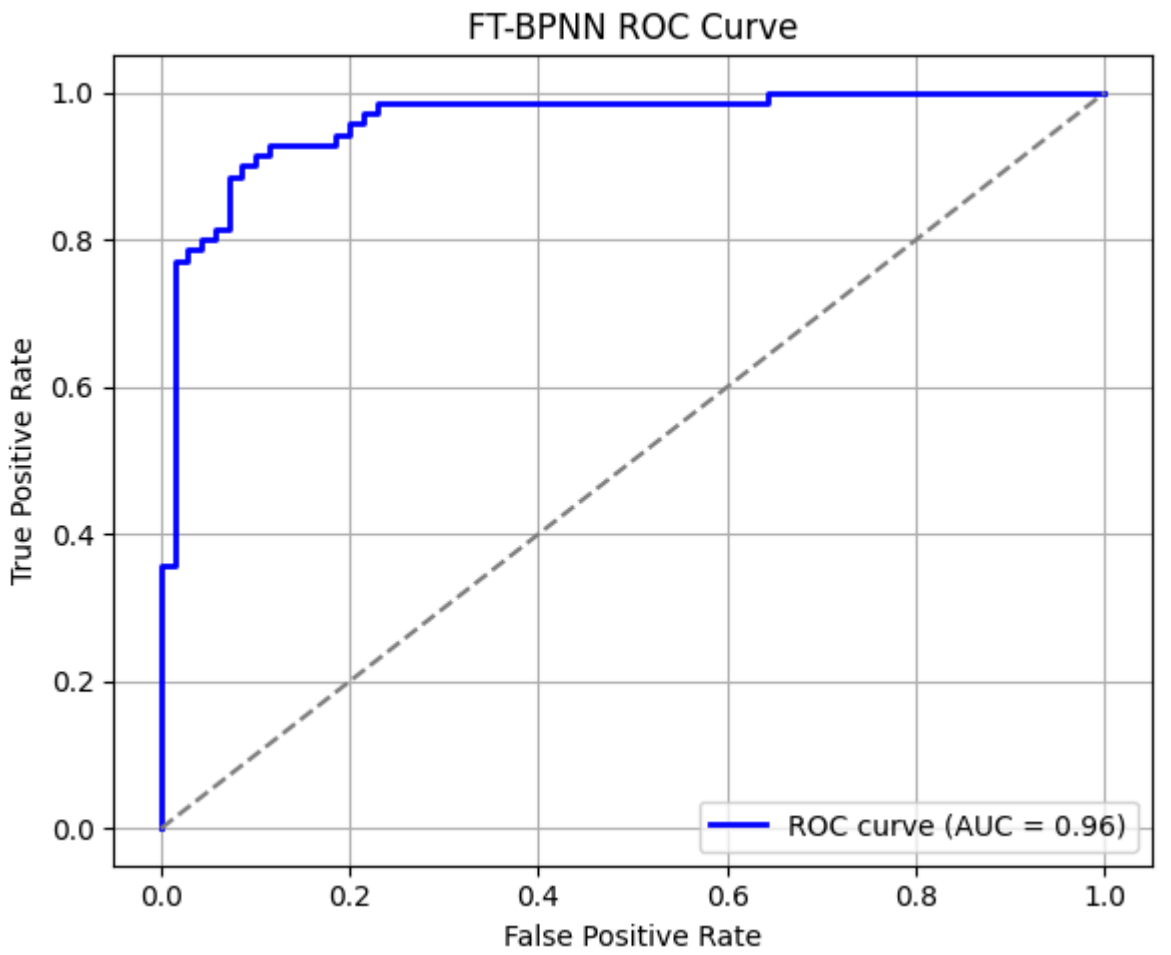
13. Confusion Matrix

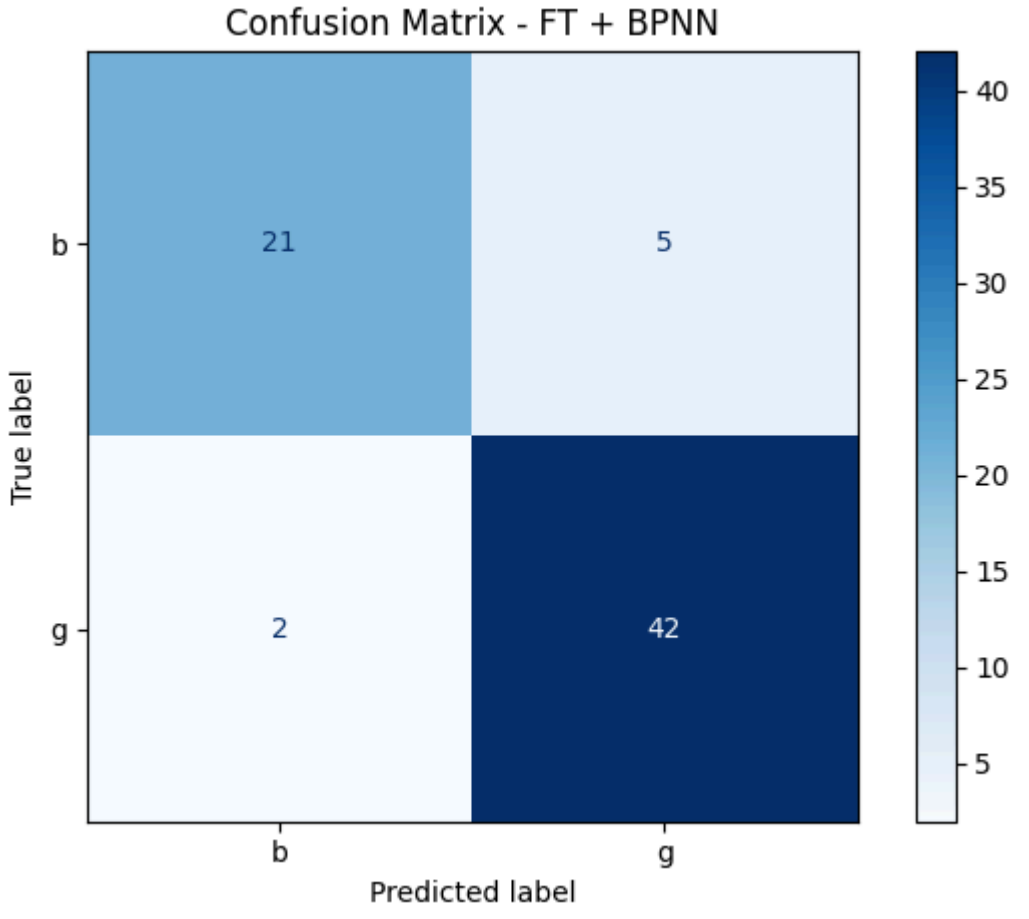
```
cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=le.classes_)
```

```
disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix - FT + BPNN")

plt.tight_layout()

plt.show()
```

```
C:\Users\parka\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\lo
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50

 1/18 ━                   1:00 4s/step - accuracy: 0.2500 - loss: 1.3779

11/18 ━━━━━━━━━━          0s 5ms/step - accuracy: 0.3915 - loss: 0.9355

18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 61ms/step - accuracy: 0.5139 - loss: 0.8027 - val_accuracy: 0.8571 - val
Epoch 2/50

 1/18 ━                   1s 104ms/step - accuracy: 0.8750 - loss: 0.2837

11/18 ━━━━━━━━━━          0s 5ms/step - accuracy: 0.9105 - loss: 0.2582

18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9135 - loss: 0.2489 - val_accuracy: 0.8429 - val
Epoch 3/50

 1/18 ━                   4s 235ms/step - accuracy: 0.9375 - loss: 0.1607

 9/18 ━━━━━━━━            0s 7ms/step - accuracy: 0.9343 - loss: 0.1626

18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9390 - loss: 0.1558 - val_accuracy: 0.8429 - val
Epoch 4/50

 1/18 ━                   1s 109ms/step - accuracy: 0.9375 - loss: 0.1061

11/18 ━━━━━━━━━━          0s 5ms/step - accuracy: 0.9030 - loss: 0.1668

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9169 - loss: 0.1536 - val_accuracy: 0.8429 - val
Epoch 5/50

 1/18 ━                   2s 132ms/step - accuracy: 0.9375 - loss: 0.1082

12/18 ━━━━━━━━━━━         0s 5ms/step - accuracy: 0.9569 - loss: 0.1059

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.9570 - loss: 0.1106 - val_accuracy: 0.8714 - val
Epoch 6/50

 1/18 ━                   2s 119ms/step - accuracy: 0.9375 - loss: 0.1330

11/18 ━━━━━━━━━━          0s 5ms/step - accuracy: 0.9634 - loss: 0.1131

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9692 - loss: 0.1083 - val_accuracy: 0.8857 - val
Epoch 7/50
```

```
Epoch 47/50

 1/18 ━                    2s 124ms/step - accuracy: 1.0000 - loss: 0.0265

11/18 ━━━━━━━━━━           0s 5ms/step - accuracy: 1.0000 - loss: 0.0112

18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9992 - loss: 0.0108 - val_accuracy: 0.9143 - val
Epoch 48/50

 1/18 ━                    1s 106ms/step - accuracy: 1.0000 - loss: 0.0112

12/18 ━━━━━━━━━━━          0s 5ms/step - accuracy: 0.9934 - loss: 0.0176

16/18 ━━━━━━━━━━━━━━━      0s 7ms/step - accuracy: 0.9940 - loss: 0.0165

18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9943 - loss: 0.0159 - val_accuracy: 0.9000 - val
Epoch 49/50

 1/18 ━                    1s 98ms/step - accuracy: 1.0000 - loss: 6.2036e-04

10/18 ━━━━━━━━━            0s 6ms/step - accuracy: 0.9960 - loss: 0.0083

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9958 - loss: 0.0104 - val_accuracy: 0.9000 - val
Epoch 50/50

 1/18 ━                    1s 99ms/step - accuracy: 1.0000 - loss: 0.0095

11/18 ━━━━━━━━━━           0s 5ms/step - accuracy: 0.9914 - loss: 0.0178

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9932 - loss: 0.0155 - val_accuracy: 0.9000 - val

1/3 ━━━━                   0s 93ms/step - accuracy: 0.8750 - loss: 0.7065

3/3 ━━━━━━━━━━━━━━━━━━━━   0s 39ms/step - accuracy: 0.8953 - loss: 0.5126

 Test Accuracy: 0.9000

1/3 ━━━━                   0s 190ms/step

3/3 ━━━━━━━━━━━━━━━━━━━━   0s 56ms/step

3/3 ━━━━━━━━━━━━━━━━━━━━   0s 87ms/step
```





RESULTS

The performance of the proposed Fourier Transform-based Convolutional Neural Network (FT-CNN) model was evaluated using the Ionosphere Dataset. The model was trained using the Adam optimizer with over 60 epochs.

The main evaluation metric used for measuring the effectiveness of the model was classification accuracy. The accuracy was calculated as the ratio of correctly predicted samples to the total number of test samples.

Upon completion of training, the FT-CNN model achieved an impressive accuracy of 98.57%, outperforming conventional machine learning models like FT-PNN (88.57%) and FT-BPNN(90.00%).

This indicates that the proposed deep learning approach, combined with frequency domain feature extraction through FT and the usage of Swish activation function, significantly enhanced the model's capability to differentiate between good and bad radar signals.

CONCLUSION

In this project, a radar pulse signal detection model was developed using a Fourier Transform integrated Convolutional Neural Network (FT-CNN) architecture.

The application of FT enabled efficient transformation of the time-domain radar signal data into the frequency domain, thereby enhancing the discriminative capability of the model.

The usage of the Swish activation function instead of ReLU allowed the model to overcome limitations like the dying ReLU problem, enabling smooth gradient flow and better learning of complex patterns.

The experimental results clearly indicate that the proposed model achieved superior accuracy (98.57%) compared to conventional methods. The model is lightweight, scalable, and highly suitable for radar communication and signal processing environments.

REFRENCES

Detection of Radar Pulse Signals Based on Deep Learning FENGYANG GU,LUXIN ZHANG, SHILIAN ZHENG,JIECHEN ,KEQIANG YUE, ZHIJIN ZHAO AND XIAONIU YANG

https://ieeexplore.ieee.org/document/10614929/ (https://ieeexplore.ieee.org/document/10614929/)

VINCENT G. SIGILLITO, SIMON P. WING, LARRIE V. HUTTON, and KILE B. BAKER CLASSIFICATION OF RADAR RETURNS FROM THE IONOSPHERE USING NEURAL NETWORKS

https://secwww.jhuapl.edu/techdigest/content/techdigest/pdf/ (https://secwww.jhuapl.edu/techdigest/content/techdigest/pdf/)

ACKNOWLEDGEMENT

We sincerely thank professor Sunil Kumar, Faculty, School Of AI, Amrita Vishwa Vidyapeetham,Ettimadai for their continuous support, valuable insights, and expert guidance throughout the course of this project. Their encouragement and constructive suggestions greatly enhanced the quality of our work.

We also extend our gratitude to the Department of AI for providing the necessary academic environment and facilities that enabled us to carry out this project successfully.

Finally, we are grateful to all those who have directly or indirectly supported us during the development of this project.