A hybrid model could involve using traditional machine learning algorithms for feature engineering or dimensionality reduction, followed by a deep learning model for more complex pattern recognition or representation learning.

Alternatively, it could involve incorporating deep learning layers within a traditional machine learning pipeline.

In this example:

1. We first train a Random Forest classifier on the reduced data obtained through PCA.
2. We then train a simple neural network on the original standardized data.
3. Finally, we create an ensemble model by combining the predictions from both the Random Forest and Neural Network.

```
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# Load the Iris dataset
iris = load_iris()
data = iris.data
target = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_standardized = scaler.fit_transform(X_train)
X_test_standardized = scaler.transform(X_test)
```

PCA reduction

```
# Apply PCA for dimensionality reduction
num_components = 2
pca = PCA(n_components=num_components)
X_train_pca = pca.fit_transform(X_train_standardized)
X_test_pca = pca.transform(X_test_standardized)
print("Standarized train PCA data:", X_train_pca[:2, :])
print("      ")
print("Standarized test PCA data:", X_test_pca[:2, :])
print("      ")
```

```
Standarized train PCA data: [[-0.16752092  1.52763072]
 [ 1.02089353  0.2868919 ]]

Standarized test PCA data: [[ 0.5226002   0.37959608]
 [-1.94983374 -1.69481837]]
```

Random Forest Classifier

```
# Train a Random Forest classifier on the reduced data
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_pca, y_train)

# Make predictions on the test set
y_pred_rf = clf.predict(X_test_pca)

# Evaluate the accuracy of the Random Forest classifier
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy of Random Forest Classifier: {accuracy_rf:.2f}')
print("        ")
```

```
Accuracy of Random Forest Classifier: 0.96
```

Neural Network Architecture

```
# Convert target labels to one-hot encoding for neural network
encoder = OneHotEncoder(sparse=False)
y_train_encoded = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_encoded = encoder.transform(y_test.reshape(-1, 1))

# Build a simple neural network
model = Sequential()
model.add(Dense(16, input_dim=num_components, activation='relu'))
model.add(Dense(3, activation='softmax'))  # Output layer with 3 neurons for Iris dataset classes
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print("Model Architecture:", model.summary())
print("        ")


# Train the neural network on the reduced data obtained through PCA
model.fit(X_train_pca, y_train_encoded, epochs=50, batch_size=8, verbose=0)

# Evaluate the neural network on the test set
_, accuracy_nn = model.evaluate(X_test_pca, y_test_encoded)
print(f'Accuracy of Neural Network: {accuracy_nn:.2f}')
print("        ")
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 16)                48

 dense_5 (Dense)             (None, 3)                 51

=================================================================
Total params: 99 (396.00 Byte)
Trainable params: 99 (396.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model Architecture: None

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
2/2 [==============================] - 0s 7ms/step - loss: 0.3174 - accuracy: 0.8444
Accuracy of Neural Network: 0.84
```

```
# Combine predictions from Random Forest and Neural Network (Ensemble)
y_pred_ensemble = np.argmax(model.predict(X_test_pca) + clf.predict_proba(X_test_pca), axis=1)
accuracy_ensemble = accuracy_score(y_test, y_pred_ensemble)
print(f'Accuracy of Ensemble Model: {accuracy_ensemble:.2f}')
```

```
2/2 [==============================] - 0s 5ms/step
Accuracy of Ensemble Model: 0.98
```

```
print(f'Accuracy of Random Forest Classifier: {accuracy_rf:.2f}')
print("        ")
print(f'Accuracy of Neural Network: {accuracy_nn:.2f}')
print("        ")
print(f'Accuracy of Ensemble Model: {accuracy_ensemble:.2f}')
```

```
Accuracy of Random Forest Classifier: 0.96

Accuracy of Neural Network: 0.84

Accuracy of Ensemble Model: 0.98
```

```
# Visualize the distribution of points in the reduced feature space
plt.figure(figsize=(12, 6))

# Plot original data before PCA
plt.subplot(1, 2, 1)
for label in np.unique(y_test):
    indices = y_test == label
    plt.scatter(X_test[indices, 0], X_test[indices, 1], label=f'Class {label}', alpha=0.7)
plt.title('Original Data Distribution')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
plt.legend()

# Plot points with true labels
plt.subplot(1, 2, 2)
for label in np.unique(y_test):
    indices = y_test == label
    plt.scatter(X_test_pca[indices, 0], X_test_pca[indices, 1], label=f'Class {label}', alpha=0.7)

# Plot points with ensemble predictions
plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred_ensemble, marker='X', s=100, cmap='viridis', edgecolor='k', label='Ensemble Pred

plt.title('Distribution of Points After PCA and Ensemble Predictions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()

plt.show()
```