Below is a revised 10-slide presentation outline that is focused primarily on the **Custom Neural Network Model** for housing price prediction. This outline includes key code snippets, evaluation metrics, and deployment steps that you can discuss during your presentation.

Slide 1: Title Slide

- **Title:** Housing Price Prediction with a Custom Neural Network
- **Subtitle:** Leveraging Deep Learning & ONNX for Mobile Deployment
- **Presented by:** *Your Name/Team*
- **Date:** *Presentation Date*
- *(Include a relevant image or your organization's logo.)*

Slide 2: Project Overview

- **Objective:**
    - Develop a custom neural network model to predict Bengaluru housing prices.
    - Deploy the trained model on a Flutter mobile app using ONNX for on-device inference.
- **Key Components:**
    - Data preparation and feature engineering.
    - Custom Neural Network (using TensorFlow/Keras).
    - Model evaluation, conversion to ONNX, and Flutter app integration.
- **Talking Points:**
    - Why deep learning? Flexibility and the ability to capture complex non-linear relationships.
    - Overview of the complete pipeline from training to mobile deployment.

Slide 3: Data and Preparation

- **Dataset:** Bengaluru House Data from Kaggle
- **Key Features:** `total_sqft`, `bath`, `balcony`, `size`
- **Target Variable:** `price`
    - **Code Snippet – Data Cleaning & Feature Extraction:**

```
import pandas as pd

# Load and clean data
df = pd.read_csv("ml_model/data/Bengaluru_House_Data.csv")
df = df.dropna()  # Remove missing values

# Filter numeric total_sqft values and convert to float
df = df[df['total_sqft'].apply(lambda x: str(x).replace('.',
'').isdigit())]
```

```
df['total_sqft'] = df['total_sqft'].astype(float)

# Extract number from 'size' (e.g., "2 BHK" -> 2)
if 'size' in df.columns:
    df['size'] = df['size'].str.extract(r"(\d+)").astype(float)
```

- **Talking Points:**
  - Emphasize the importance of cleaning and preprocessing.
  - Discuss how proper feature engineering leads to better model performance.

Slide 4: Custom Neural Network Architecture

- **Model Overview:**
  - A fully connected feedforward neural network built with TensorFlow/Keras.
- **Architecture Details:**
  - **Input Layer:** Matches the number of features (4).
  - **Hidden Layers:** Two Dense layers with 64 neurons each and ReLU activation.
  - **Dropout Layers:** 20% dropout after each hidden layer to prevent overfitting.
  - **Output Layer:** Single neuron with no activation for regression.
  - **Code Snippet – Building the Model:**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Hyperparameters
neurons_per_layer = 64
dropout_rate = 0.2
learning_rate = 0.001

# Build the model
model = Sequential([
    Dense(neurons_per_layer, activation='relu', input_dim=4),
    Dropout(dropout_rate),
    Dense(neurons_per_layer, activation='relu'),
    Dropout(dropout_rate),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='mse', metrics=['mae'])
```

- **Talking Points:**
  - Explain the rationale behind choosing the number of layers, neurons, and dropout.
  - How each layer contributes to learning complex non-linear patterns.

Slide 5: Model Training Process

- **Training Setup:**
    - Data is split into training and testing sets.
    - Features are scaled using `StandardScaler` for faster convergence.
    - Early stopping is implemented to avoid overfitting.

- **Code Snippet – Training the Model:**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import EarlyStopping

# Prepare features and target
features = ['total_sqft', 'bath', 'balcony', 'size']
target = 'price'

X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Set up early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.1,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

- **Talking Points:**
    - Discuss the importance of early stopping and validation split.
    - Mention how scaling impacts the performance of deep neural networks.

Slide 6: Model Evaluation

- **Evaluation Metrics:**

- **Mean Absolute Error (MAE):** Provides the average magnitude of prediction errors.
- **Root Mean Squared Error (RMSE):** Gives more weight to larger errors.

- **Code Snippet – Evaluating the Model:**

```
import numpy as np
from sklearn.metrics import mean_absolute_error,
mean_squared_error

# Generate predictions
y_train_pred = model.predict(X_train_scaled).flatten()
y_test_pred = model.predict(X_test_scaled).flatten()

# Calculate evaluation metrics
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

print(f"Train MAE: {train_mae:.2f}, Test MAE: {test_mae:.2f}")
print(f"Train RMSE: {train_rmse:.2f}, Test RMSE:
{test_rmse:.2f}")
```

- **Sample Values (Hypothetical):**

  - **Train MAE:** 28.5, **Test MAE:** 35.2
  - **Train RMSE:** 40.1, **Test RMSE:** 48.3

- **Talking Points:**

  - Explain what MAE and RMSE reveal about the model's performance.
  - Discuss potential overfitting or underfitting if training and testing metrics differ significantly.

Slide 7: Converting the Custom Model to ONNX

- **Why Convert to ONNX?**

  - Enables cross-platform and on-device inference in mobile apps.

- **Conversion Process:**

  - Use **tf2onnx** to convert the TensorFlow/Keras model to ONNX format.

    - **Code Snippet – Converting with tf2onnx:**

```
# From the command line, run:
python -m tf2onnx.convert --saved-model path/to/saved_model --
output house_price_model.onnx
```

    - *Or via Python:*

```
import tf2onnx
import tensorflow as tf

# Save the Keras model
model.save("custom_nn_house_price_model")

# Convert to ONNX
spec = (tf.TensorSpec((None, 4), tf.float32, name="input"),)
output_path = "ml_model/house_price_model.onnx"
model_proto, _ = tf2onnx.convert.from_keras(model,
input_signature=spec, output_path=output_path)
print("Custom NN model converted to ONNX.")
```

- **Talking Points:**
  - Emphasize the ease of conversion and benefits of ONNX for deployment.
  - Mention that the resulting ONNX model is used in the Flutter app.

Slide 8: Flutter App Integration

- **App Functionality:**
  - The Flutter app loads the ONNX model for real-time prediction.
  - **Dart Code Snippet (Pseudo-code):**

```
import 'package:onnxruntime/onnxruntime.dart';

Future<double> predictHousePrice(List<double> features) async {
  // Load the ONNX model from assets
  final session = await
OrtSession.fromAsset('assets/house_price_model.onnx');

  // Prepare input tensor (ensure shape is [1, 4])
  final inputTensor = OrtTensor.fromList(features, shape: [1,
4]);

  // Run inference
  final outputs = await session.run({'input': inputTensor});
  final prediction = outputs['output_0']?.first;
  return prediction ?? 0.0;
}
```

- **Talking Points:**
  - Explain how ONNX Runtime enables seamless integration with mobile platforms.
  - Walk through the steps: model loading, input preparation, inference, and output retrieval.

Slide 9: Project Architecture & Workflow

- **High-Level Workflow Diagram:**

```
[Data Collection & Preprocessing]
          ↓
[Custom NN Model Training & Evaluation]
          ↓
[Model Conversion to ONNX]
          ↓
[Flutter App Integration]
          ↓
[On-device Prediction]
```

- **Talking Points:**
    - Explain each step in the pipeline.
    - Highlight the modularity of the system and how each component interconnects.

Slide 10: Summary, Future Work & Q&A

- **Project Recap:**
    - Developed a custom neural network for housing price prediction.
    - Evaluated the model using MAE and RMSE, achieving competitive performance.
    - Converted the model to ONNX for mobile deployment in a Flutter app.
- **Future Directions:**
    - Experiment with deeper architectures and additional features.
    - Optimize the neural network hyperparameters further.
    - Enhance the Flutter app UI/UX and add more functionalities.
- **Q&A:**
    - Open the floor for questions and feedback.
- *(Conclude with a thank-you message and your contact information.)*

This presentation outline focuses on the custom neural network model—from data preparation and model architecture to training, evaluation, and eventual deployment via ONNX in a Flutter app. Each slide includes code snippets and key values or metrics to help illustrate your process and results. Feel free to adjust the content as needed to suit your audience and project specifics. Enjoy your presentation!

Below is the full text content for a 10-slide presentation focused on your custom neural network model. You can copy and paste this content into your preferred

presentation tool (PowerPoint, Google Slides, etc.) and adjust the formatting or add visuals as needed.

Slide 1: Title Slide

Housing Price Prediction with a Custom Neural Network
Leveraging Deep Learning & ONNX for Mobile Deployment

Presented by: Your Name/Team
Date: [Presentation Date]

(Add an appropriate background image or your organization's logo.)

Slide 2: Project Overview

Objective:

- Develop a custom neural network model to predict housing prices in Bengaluru.
- Deploy the trained model on a Flutter mobile app using ONNX for on-device inference.

Key Components:

- **Data Preparation & Feature Engineering:** Cleaning and processing the raw housing data.
- **Custom Neural Network:** Building a deep learning model with TensorFlow/Keras.
- **Model Evaluation:** Assessing performance with MAE and RMSE.
- **ONNX Conversion:** Converting the trained model to ONNX for cross-platform compatibility.
- **Flutter Integration:** Deploying the model within a mobile application for real-time predictions.

Talking Points:

- Why deep learning? Its flexibility to capture complex, non-linear relationships.
- Overview of our end-to-end pipeline—from data to mobile deployment.

Slide 3: Data and Preparation

Dataset:

- Sourced from Kaggle: Bengaluru House Data
- **Key Features:** `total_sqft`, `bath`, `balcony`, `size`
- **Target Variable:** `price`

Data Preparation Steps:

- Removing missing values.

- Converting `total_sqft` to a numeric format.
- Extracting numeric values from the `size` column.

Code Snippet:

```
import pandas as pd

# Load and clean the dataset
df = pd.read_csv("ml_model/data/Bengaluru_House_Data.csv")
df = df.dropna()  # Remove missing values

# Ensure 'total_sqft' is numeric
df = df[df['total_sqft'].apply(lambda x: str(x).replace('.',
'').isdigit())]
df['total_sqft'] = df['total_sqft'].astype(float)

# Extract number from 'size' (e.g., "2 BHK" -> 2)
if 'size' in df.columns:
    df['size'] = df['size'].str.extract(r"(\d+)").astype(float)
```

Talking Points:

- The importance of data cleaning and feature extraction in improving model performance.

Slide 4: Custom Neural Network Architecture

Model Overview:
We designed a fully connected feedforward neural network with the following architecture:

- **Input Layer:** 4 neurons (one for each feature).
- **Hidden Layers:** Two Dense layers with 64 neurons each and ReLU activation.
- **Dropout Layers:** 20% dropout after each hidden layer to reduce overfitting.
- **Output Layer:** 1 neuron (for regression output).

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Hyperparameters
neurons_per_layer = 64
dropout_rate = 0.2
learning_rate = 0.001

# Build the model
model = Sequential([
    Dense(neurons_per_layer, activation='relu', input_dim=4),
    Dropout(dropout_rate),
    Dense(neurons_per_layer, activation='relu'),
    Dropout(dropout_rate),
    Dense(1)  # Regression output
])
```

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mse',
metrics=['mae'])
```

Talking Points:

- Why we chose two hidden layers and a 20% dropout rate.
- How this architecture helps capture complex non-linear relationships in the data.

Slide 5: Model Training Process

Training Setup:

- **Data Splitting:** Dividing data into training and test sets.
- **Feature Scaling:** Using `StandardScaler` to standardize feature values.
- **Early Stopping:** Monitoring validation loss to prevent overfitting.

Code Snippet:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import EarlyStopping

# Define features and target
features = ['total_sqft', 'bath', 'balcony', 'size']
target = 'price'
X = df[features]
y = df[target]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.1,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

Talking Points:

- The role of scaling and early stopping in improving convergence and model generalization.

Slide 6: Model Evaluation

Evaluation Metrics:

- **Mean Absolute Error (MAE):** Average magnitude of errors.
- **Root Mean Squared Error (RMSE):** Gives extra weight to larger errors.

Code Snippet:

```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Generate predictions
y_train_pred = model.predict(X_train_scaled).flatten()
y_test_pred = model.predict(X_test_scaled).flatten()

# Calculate evaluation metrics
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

print(f"Train MAE: {train_mae:.2f}, Test MAE: {test_mae:.2f}")
print(f"Train RMSE: {train_rmse:.2f}, Test RMSE: {test_rmse:.2f}")
```

Hypothetical Results:

- **Train MAE:** 28.5, **Test MAE:** 35.2
- **Train RMSE:** 40.1, **Test RMSE:** 48.3

Talking Points:

- What these metrics indicate about model performance.
- Possible signs of overfitting if the training error is significantly lower than the testing error.

Slide 7: Converting the Custom Model to ONNX

Why Convert to ONNX?

- Enables cross-platform deployment and efficient on-device inference in mobile apps.

Conversion Process:

- We use **tf2onnx** to convert our TensorFlow/Keras model.

Code Snippet (Using Python):

```
import tf2onnx
import tensorflow as tf

# Save the Keras model in SavedModel format
model.save("custom_nn_house_price_model")

# Define input signature: batch size is None and 4 features per
sample
spec = (tf.TensorSpec((None, 4), tf.float32, name="input"),)
output_path = "ml_model/house_price_model.onnx"

# Convert the SavedModel to ONNX
model_proto, _ = tf2onnx.convert.from_keras(model,
input_signature=spec, output_path=output_path)
print("Custom NN model converted to ONNX.")
```

Talking Points:

- The simplicity of converting a Keras model to ONNX.
- How ONNX format ensures our model can be used in different runtime environments, such as in Flutter.

Slide 8: Flutter App Integration

App Functionality:

- The Flutter app loads the ONNX model and uses ONNX Runtime for predictions.

Dart Code Snippet (Pseudo-code):

```
import 'package:onnxruntime/onnxruntime.dart';

Future<double> predictHousePrice(List<double> features) async {
  // Load the ONNX model from assets
  final session = await
OrtSession.fromAsset('assets/house_price_model.onnx');

  // Prepare the input tensor (ensure shape is [1, 4])
  final inputTensor = OrtTensor.fromList(features, shape: [1, 4]);

  // Run inference
  final outputs = await session.run({'input': inputTensor});

  // Retrieve and return the prediction
  final prediction = outputs['output_0']?.first;
  return prediction ?? 0.0;
}
```

Talking Points:

- How the ONNX model is integrated into the Flutter app.
- The steps involved: loading the model, preparing the input, running inference, and retrieving the prediction.

Slide 9: Project Architecture & Workflow

Workflow Diagram:

```
[Data Collection & Preprocessing]
            ↓
[Custom Neural Network Model Training & Evaluation]
            ↓
[Model Conversion to ONNX]
            ↓
[Flutter App Integration]
            ↓
[On-device Prediction]
```

Talking Points:

- Explain each stage of the pipeline.
- Emphasize the modular design that allows improvements in any stage (e.g., data preprocessing, model tuning, or app integration).

Slide 10: Summary, Future Work & Q&A

Project Recap:

- Developed a custom neural network to predict housing prices.
- Achieved competitive performance with MAE and RMSE metrics.
- Converted the model to ONNX and integrated it into a Flutter app for real-time, on-device predictions.

Future Directions:

- Experiment with deeper architectures and additional features.
- Further tune hyperparameters to improve performance.
- Enhance the Flutter app UI/UX and add more functionalities.

Q&A:

- Thank you for your attention! I welcome any questions or feedback.

(Include your contact information and a "Thank You" note.)

This content is designed to guide you through a detailed presentation centered on your custom neural network model—from data preparation and model design to evaluation, ONNX conversion, and mobile deployment with Flutter. Feel free to modify and expand upon these slides as needed for your audience. Enjoy your presentation!