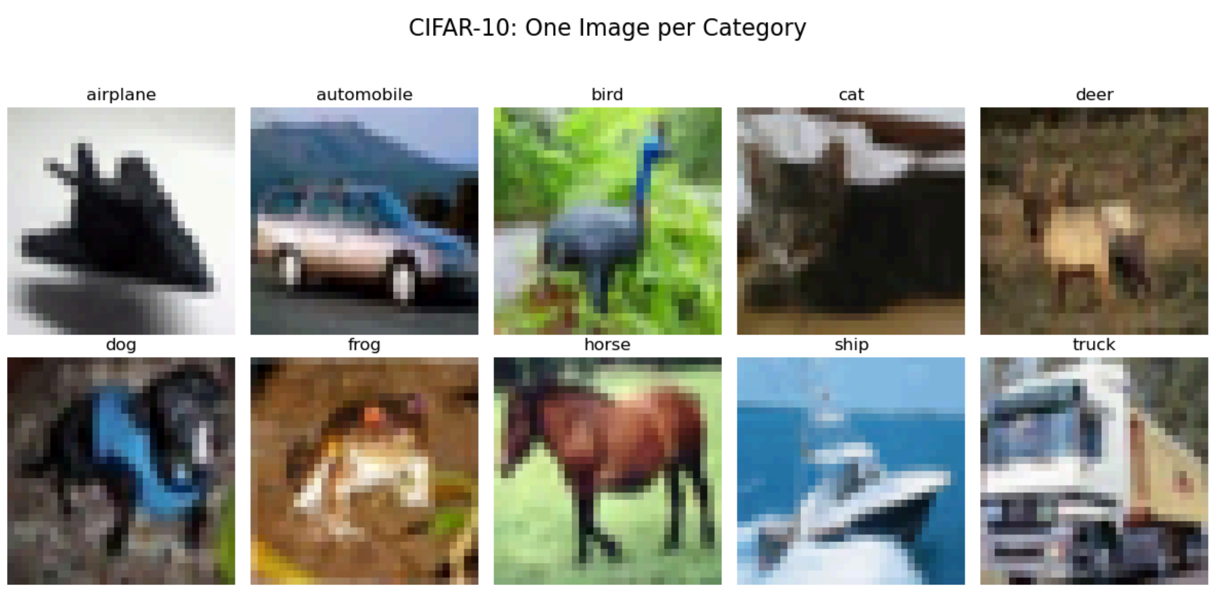
In [ ]:
```python
#Tom Deng
#662007936
```

In [1]:
```python
#Problem 1a
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Define the class names for CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Plot one image per category from the training set
plt.figure(figsize=(12, 6))

for i in range(10):
    # Find the first index where the label equals i
    idx = np.where(y_train.flatten() == i)[0][0]
    image = x_train[idx]

    # Create a subplot: 2 rows x 5 columns
    plt.subplot(2, 5, i + 1)
    plt.imshow(image)
    plt.title(class_names[i])
    plt.axis('off')

plt.suptitle("CIFAR-10: One Image per Category", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



In [5]:
```python
#Problem 1b
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
```

Loading [MathJax]/extensions/Safe.js

```python
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier  # Use SciKeras instead of the
from sklearn.model_selection import GridSearchCV, train_test_split

# Load the CIFAR-10 dataset
(x_full, y_full), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize pixel values to [0, 1]
x_full = x_full.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Use a 70%-30% split for training and validation data
x_train, x_val, y_train, y_val = train_test_split(x_full, y_full, test_size=
y_train = y_train.flatten()  # Flatten labels from shape (n,1) to (n,)
y_val = y_val.flatten()

# Define a function to create the neural network model.
def create_model(neurons=128, dropout_rate=0.0, learning_rate=0.001):
    model = Sequential()
    # Input layer: flatten CIFAR-10 images of shape (32,32,3)
    model.add(Flatten(input_shape=(32, 32, 3)))
    # Hidden dense layer with a variable number of neurons and ReLU activati
    model.add(Dense(neurons, activation='relu'))
    # Optional dropout layer for regularization
    if dropout_rate > 0:
        model.add(Dropout(dropout_rate))
    # Output layer for 10 classes with softmax activation
    model.add(Dense(10, activation='softmax'))

    # Compile the model using Adam optimizer with the specified learning rat
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Wrap the model using SciKeras's KerasClassifier
model_wrapper = KerasClassifier(model=create_model, verbose=0)

# Define the grid of hyperparameters to search over.
# Note: When using SciKeras, the hyperparameters for the underlying model ne
param_grid = {
    'model__neurons': [64, 128],
    'model__dropout_rate': [0.0, 0.2],
    'model__learning_rate': [0.001, 0.01],
    'batch_size': [32, 64],
    'epochs': [10]  # Using 10 epochs for demonstration purposes
}

# Set up the GridSearchCV object using 3-fold cross-validation
grid = GridSearchCV(estimator=model_wrapper, param_grid=param_grid, cv=3)

# Perform grid search on the training set
grid_result = grid.fit(x_train, y_train)
```

```
# Print the best hyperparameters and the corresponding accuracy
print("Best Accuracy: {:.4f} using {}".format(grid_result.best_score_, grid_
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[5], line 57
     54 grid = GridSearchCV(estimator=model_wrapper, param_grid=param_grid, 
cv=3)
     56 # Perform grid search on the training set
---> 57 grid_result = grid.fit(x_train, y_train)
     59 # Print the best hyperparameters and the corresponding accuracy
     60 print("Best Accuracy: {:.4f} using {}".format(grid_result.best_score
_, grid_result.best_params_))

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1389, in _fit_context.<lo
cals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1382     estimator._validate_params()
   1384 with config_context(
   1385     skip_parameter_validation=(
   1386         prefer_skip_nested_validation or global_skip_validation
   1387     )
   1388 ):
-> 1389     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:933, i
n BaseSearchCV.fit(self, X, y, **params)
    929 params = _check_method_params(X, params=params)
    931 routed_params = self._get_routed_params_for_fit(params)
--> 933 cv_orig = check_cv(self.cv, y, classifier=is_classifier(estimator))
    934 n_splits = cv_orig.get_n_splits(X, y, **routed_params.splitter.spli
t)
    936 base_estimator = clone(self.estimator)

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1237, in is_classifier(es
timator)
   1230     warnings.warn(
   1231         f"passing a class to {print(inspect.stack()[0][3])} is depre
cated and "
   1232         "will be removed in 1.8. Use an instance of the class instea
d.",
   1233         FutureWarning,
   1234     )
   1235     return getattr(estimator, "_estimator_type", None) == "classifie
r"
-> 1237 return get_tags(estimator).estimator_type == "classifier"

File ~\anaconda3\Lib\site-packages\sklearn\utils\_tags.py:430, in get_tags(e
stimator)
    428 for klass in reversed(type(estimator).mro()):
    429     if "__sklearn_tags__" in vars(klass):
--> 430         sklearn_tags_provider[klass] = klass.__sklearn_tags__(estima
tor)  # type: ignore[attr-defined]
    431         class_order.append(klass)
    432     elif "_more_tags" in vars(klass):

File ~\anaconda3\Lib\site-packages\sklearn\base.py:540, in ClassifierMixin._
_sklearn_tags__(self)
    539 def __sklearn_tags__(self):
    540     tags = super().__sklearn_tags__()
```

Loading [MathJax]/extensions/Safe.js

```
    541     tags.estimator_type = "classifier"
    542     tags.classifier_tags = ClassifierTags()

AttributeError: 'super' object has no attribute '__sklearn_tags__'
```

In [6]:

```
#Problem 1 c

# Refit the best estimator on the training set with validation data
# (This step is done to capture the training history on the full training se
# Note: best_model is the best estimator from grid search.
best_model = grid_result.best_estimator_

# Refit using the training (x_train, y_train) and validation (x_val, y_val)
# We use the same number of epochs that were selected by the grid search.
best_epochs = best_model.get_params()['epochs']
best_model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=best

# Retrieve the training history from the SciKeras wrapper (stored in best_mo
history_tuned = best_model.history_

# Extract loss values from the history
train_loss = history_tuned['loss']
val_loss = history_tuned['val_loss']
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation losses over epochs
plt.figure(figsize=(8, 6))
plt.plot(epochs, train_loss, 'bo-', label='Training Loss')
plt.plot(epochs, val_loss, 'ro-', label='Validation Loss')
plt.title('Training and Validation Losses of the Tuned Network')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 6
      1 #Problem 1 c
      2
      3 # Refit the best estimator on the training set with validation data
      4 # (This step is done to capture the training history on the full tra
ining set with our separate validation set)
      5 # Note: best_model is the best estimator from grid search.
----> 6 best_model = grid_result.best_estimator_
      8 # Refit using the training (x_train, y_train) and validation (x_val,
y_val) sets.
      9 # We use the same number of epochs that were selected by the grid se
arch.
     10 best_epochs = best_model.get_params()['epochs']

NameError: name 'grid_result' is not defined
```

In [8]:

```
#problem 1d
# Extract training and validation accuracies from the history of the best mo
train_accuracy = best_model.history_['accuracy']
```

Loading [MathJax]/extensions/Safe.js

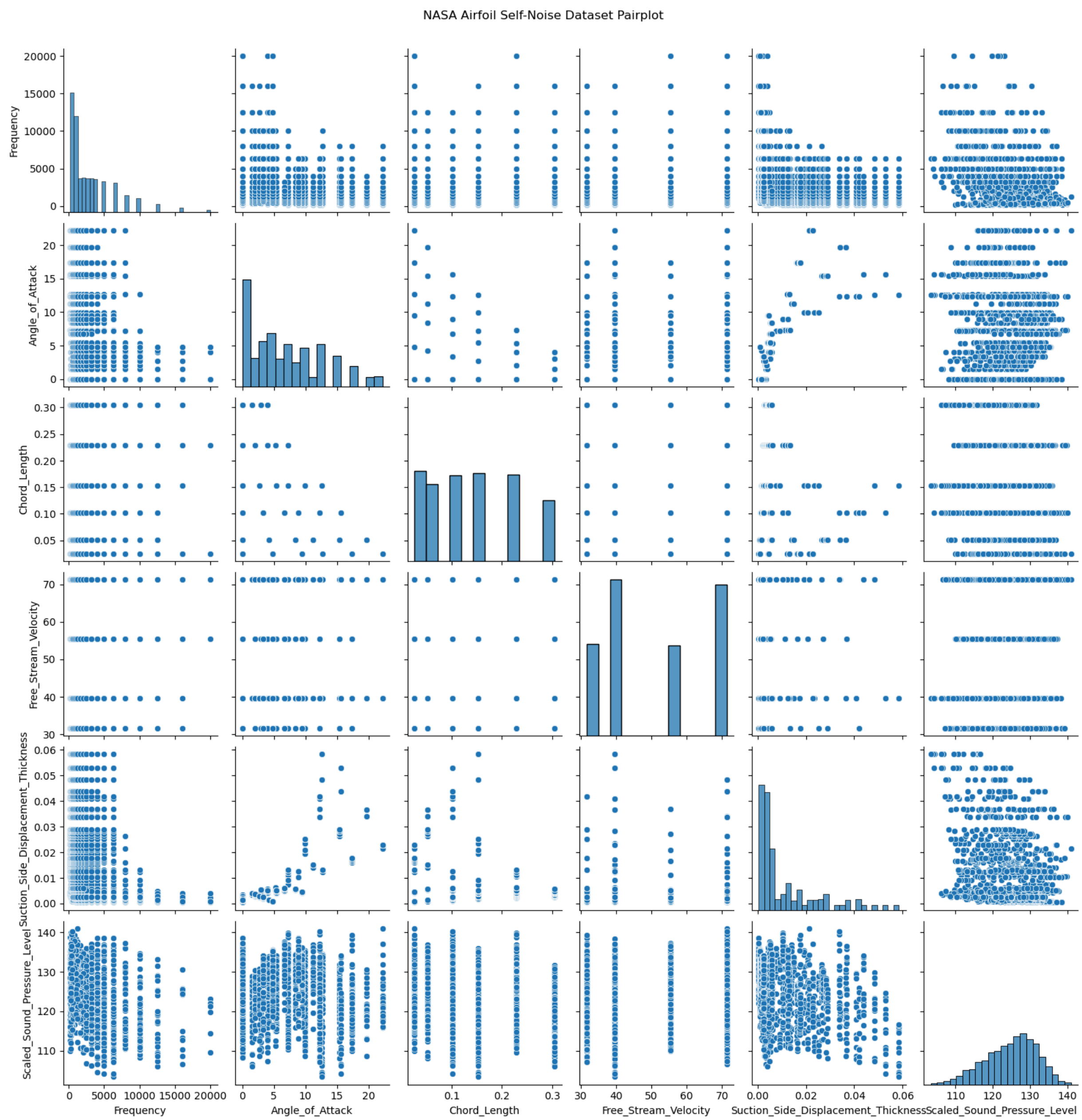
```
val_accuracy = best_model.history_['val_accuracy']
epochs = range(1, len(train_accuracy) + 1)

# Plot the training and validation accuracies over epochs
plt.figure(figsize=(8, 6))
plt.plot(epochs, train_accuracy, 'bo-', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'ro-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy of the Tuned Network')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 3
      1 #problem 1d
      2 # Extract training and validation accuracies from the history of the
best model
----> 3 train_accuracy = best_model.history_['accuracy']
      4 val_accuracy = best_model.history_['val_accuracy']
      5 epochs = range(1, len(train_accuracy) + 1)

NameError: name 'best_model' is not defined
```

In [9]:

```
#Problem 2 a
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# URL for the dataset from the UCI repository
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoi

# Define the column names:
# 1. Frequency (Hz)
# 2. Angle of attack (degrees)
# 3. Chord length (m)
# 4. Free-stream velocity (m/s)
# 5. Suction side displacement thickness (m)
# 6. Scaled sound pressure level (dB)
columns = ["Frequency", "Angle_of_Attack", "Chord_Length",
           "Free_Stream_Velocity", "Suction_Side_Displacement_Thickness",
           "Scaled_Sound_Pressure_Level"]

# Load the dataset (the file is whitespace-delimited)
airfoil_data = pd.read_csv(url, delim_whitespace=True, names=columns)

# Display the first few rows for verification
print(airfoil_data.head())

# Visualize the dataset using a pairplot to see the distributions and pairwi
sns.pairplot(airfoil_data)
plt.suptitle("NASA Airfoil Self-Noise Dataset Pairplot", y=1.02)
plt.show()
```

```
C:\Users\Tom\AppData\Local\Temp\ipykernel_14268\1723980045.py:21: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a future version. Use ``sep='\s+'`` instead
  airfoil_data = pd.read_csv(url, delim_whitespace=True, names=columns)
   Frequency  Angle_of_Attack  Chord_Length  Free_Stream_Velocity  \
0        800              0.0        0.3048                  71.3
1       1000              0.0        0.3048                  71.3
2       1250              0.0        0.3048                  71.3
3       1600              0.0        0.3048                  71.3
4       2000              0.0        0.3048                  71.3

   Suction_Side_Displacement_Thickness  Scaled_Sound_Pressure_Level
0                             0.002663                      126.201
1                             0.002663                      125.201
2                             0.002663                      125.951
3                             0.002663                      127.591
4                             0.002663                      127.461
```



Loading [MathJax]/extensions/Safe.js

In [11]:

```
#Problem 2b
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf

# Separate the features (X) and the target (y)
X = airfoil_data.drop("Scaled_Sound_Pressure_Level", axis=1)
y = airfoil_data["Scaled_Sound_Pressure_Level"]

# Split the data into 70% training and 30% validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, rando

# Standardize the features (this is important for neural network training)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Build the fully connected neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(X_train_scaled.shape[1],)),  # Input layer
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer for regression (no activation,
])

# Compile the model using Mean Squared Error as the loss function and Adam o
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='mse',
              metrics=['mae'])

# Train the model; here we use 50 epochs and a batch size of 32 (adjust as n
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,
                    validation_data=(X_val_scaled, y_val))
```

```
Epoch 1/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 15622.1934 - mae: 124.7953 -
val_loss: 15388.0107 - val_mae: 123.8542
Epoch 2/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 15275.1055 - mae: 123.3839 -
val_loss: 14879.0254 - val_mae: 121.7695
Epoch 3/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 14662.0713 - mae: 120.8518 -
val_loss: 13783.1719 - val_mae: 117.1465
Epoch 4/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 13250.2578 - mae: 114.7790 -
val_loss: 11753.3105 - val_mae: 108.0165
Epoch 5/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 11169.4385 - mae: 105.0551 -
val_loss: 8712.8145 - val_mae: 92.5349
Epoch 6/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7738.0093 - mae: 86.5365 - va
l_loss: 5133.6304 - val_mae: 69.6579
Epoch 7/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4359.1558 - mae: 62.9489 - va
l_loss: 2277.8174 - val_mae: 43.4563
Epoch 8/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1980.9077 - mae: 39.2309 - va
l_loss: 937.1115 - val_mae: 25.9005
Epoch 9/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 932.9691 - mae: 25.4647 - val
_loss: 569.0662 - val_mae: 19.0789
Epoch 10/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 614.9474 - mae: 20.1019 - val
_loss: 467.9100 - val_mae: 17.1468
Epoch 11/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 438.9889 - mae: 16.8419 - val
_loss: 419.1237 - val_mae: 16.3004
Epoch 12/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 443.4260 - mae: 16.7755 - val
_loss: 387.1950 - val_mae: 15.7389
Epoch 13/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 392.7084 - mae: 15.6963 - val
_loss: 358.6152 - val_mae: 15.1716
Epoch 14/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 335.5869 - mae: 14.5912 - val
_loss: 335.8758 - val_mae: 14.6941
Epoch 15/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 342.6508 - mae: 14.7475 - val
_loss: 314.6344 - val_mae: 14.2202
Epoch 16/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 305.2450 - mae: 13.9199 - val
_loss: 293.8429 - val_mae: 13.7370
Epoch 17/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 295.6964 - mae: 13.6467 - val
_loss: 275.1071 - val_mae: 13.2918
Epoch 18/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 273.1523 - mae: 13.4087 - val
_loss: 258.2428 - val_mae: 12.8829
Epoch 19/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 271.3279 - mae: 13.1148 - val
```

```
_loss: 241.1933 - val_mae: 12.4448
Epoch 20/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 254.5111 - mae: 12.5830 - val
_loss: 226.2947 - val_mae: 12.0643
Epoch 21/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 232.5550 - mae: 12.2356 - val
_loss: 212.5589 - val_mae: 11.6853
Epoch 22/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 208.2151 - mae: 11.3450 - val
_loss: 198.7071 - val_mae: 11.2915
Epoch 23/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 197.2529 - mae: 11.3413 - val
_loss: 188.2326 - val_mae: 11.0067
Epoch 24/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 180.9360 - mae: 10.8260 - val
_loss: 178.9189 - val_mae: 10.7399
Epoch 25/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 181.3467 - mae: 10.8181 - val
_loss: 167.4049 - val_mae: 10.3825
Epoch 26/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 176.4547 - mae: 10.5005 - val
_loss: 157.9093 - val_mae: 10.0870
Epoch 27/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 149.5669 - mae: 9.8492 - val_
loss: 149.6966 - val_mae: 9.8213
Epoch 28/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 152.6105 - mae: 9.9029 - val_
loss: 142.2546 - val_mae: 9.5915
Epoch 29/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 143.4135 - mae: 9.5698 - val_
loss: 135.5409 - val_mae: 9.3585
Epoch 30/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 141.2914 - mae: 9.4797 - val_
loss: 128.0948 - val_mae: 9.1117
Epoch 31/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 118.7784 - mae: 8.7852 - val_
loss: 121.6206 - val_mae: 8.8692
Epoch 32/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 121.1519 - mae: 8.7764 - val_
loss: 115.5658 - val_mae: 8.6400
Epoch 33/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 118.9000 - mae: 8.6441 - val_
loss: 110.9659 - val_mae: 8.4807
Epoch 34/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 113.8454 - mae: 8.5699 - val_
loss: 104.7240 - val_mae: 8.2245
Epoch 35/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 104.9080 - mae: 8.2130 - val_
loss: 99.9755 - val_mae: 8.0430
Epoch 36/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 101.7054 - mae: 8.0934 - val_
loss: 94.7653 - val_mae: 7.8297
Epoch 37/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 93.4992 - mae: 7.7426 - val_l
oss: 90.0378 - val_mae: 7.6203
Epoch 38/50
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 84.2176 - mae: 7.3502 - val_l
oss: 85.9848 - val_mae: 7.4460
Epoch 39/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 84.1880 - mae: 7.2096 - val_l
oss: 81.6156 - val_mae: 7.2336
Epoch 40/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 81.2114 - mae: 7.0831 - val_l
oss: 78.2787 - val_mae: 7.0885
Epoch 41/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 76.8248 - mae: 6.9233 - val_l
oss: 74.5512 - val_mae: 6.9048
Epoch 42/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 66.2104 - mae: 6.4644 - val_l
oss: 71.3438 - val_mae: 6.7463
Epoch 43/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 73.1799 - mae: 6.7696 - val_l
oss: 68.5112 - val_mae: 6.5837
Epoch 44/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 62.8856 - mae: 6.3254 - val_l
oss: 64.5460 - val_mae: 6.3726
Epoch 45/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 62.2360 - mae: 6.2559 - val_l
oss: 62.1642 - val_mae: 6.2714
Epoch 46/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 62.1814 - mae: 6.2545 - val_l
oss: 59.0711 - val_mae: 6.0684
Epoch 47/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 55.2808 - mae: 5.9882 - val_l
oss: 57.9542 - val_mae: 5.9683
Epoch 48/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 53.1818 - mae: 5.7670 - val_l
oss: 54.5017 - val_mae: 5.8060
Epoch 49/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 51.2184 - mae: 5.6193 - val_l
oss: 53.2056 - val_mae: 5.7312
Epoch 50/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 53.1577 - mae: 5.7422 - val_l
oss: 50.8359 - val_mae: 5.6154
```

In [12]:

```
#2c)
from sklearn.metrics import r2_score

# Predict the scaled sound pressure level on the validation set
y_val_pred = model.predict(X_val_scaled).flatten()

# Calculate the coefficient of determination (R² score)
r2 = r2_score(y_val, y_val_pred)
print("Coefficient of Determination (R²) on the Validation Set:", r2)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Coefficient of Determination (R²) on the Validation Set: -0.0767439752932268
5
```

In [ ]: