source_code_aerosols-img

June 24, 2024

[ ]:
```
1. Introdução

Este relatorio para a UC de Redes Neuronais e Aprendizagem Profunda tem como␣
 ↪objetivo construir um modelo capaz de estimar a AOT (Aerosol Optical␣
 ↪Thickness) a 550 nm para uma localização específica a partir da imagem␣
 ↪correspondente dos satelites Sentinel-2 em conjunto com os dados recolhidos␣
 ↪pelas estações AERONET.


Descrição dos Dados para construir o modelo:

Imagens Sentinel-2 de 19x19 pixels centradas na localização da estação AERONET.

Medidas de elevação, ozono e NO2 obtidas a partir da estação AERONET;

Ângulos solares (azimute e zenite);

Ângulos de incidência do satélite (azimute e zenite);

Nome de cada ficheiro de imagem coreespondente, com o qual podemos identificar␣
 ↪a data, a hora da imagem e as cordenadas da estação AERONET.


Feature a prever:

value_550: AOT de 550 nm para qualquer localização com base nos dados dos␣
 ↪satelites Sentinel-2.


No geral, o dataset é composto por 13.150 pontos de dados, com 10439␣
 ↪observaçoes para treino e 2711 para teste.
```

[1]:
```
# Bibliotecas Gerais
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold
import matplotlib.pyplot as plt

# TensorFlow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2, l1
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
```

[2]:
```
# Ler dados
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

[ ]:
```
2. Pré-processamento dos Dados

As etapas de pré-processamento dos dados de treino e de teste são abrangentes, de modo a extrair o maximo de features que revelem valor na predição do valor AOT 550 nm:

Passando pelas seguintes etapas:

   Extração de features a partir do nome do ficheiro de imagem, como a data, a hora e as coordenadas da estação AERONET.

   Remoção da coluna id que não tem valor preditivo, e da coluna file_name_l1 que foi usada para extrair as features mencionadas acima.

   Ordenação dos valores de AOT 550 nm  para testar a construção do modelo com a variável target ordenada.

   Divisão dos dados de treino em train_set e validation_set, de modo a poder construir um modelo que seja capaz de generalizar para dados não vistos.

   Standardização dos dados de treino e de teste, de modo a garantir que todas as features tenham a mesma escala e não seja um fator que influencie a predição do modelo.
```

[3]:
```
# Função para dividir o nome do file e extrair informações
def extract_features_from_filename(filename):
    parts = filename.split('_')

    def clean_coordinate(coordinate):
```

```
        clean_coord = coordinate.replace('-', '')
        try:
            return float(clean_coord)
        except ValueError:
            return clean_coord


    if len(parts) == 11:
        coordinates_id1 = clean_coordinate(parts[4])
        coordinates_id2 = clean_coordinate(parts[5])
        date_time_acquisition_start = parts[8].replace('T', '')
        date_time_acquisition_end = parts[9].replace('T', '')
        return pd.Series([coordinates_id1, coordinates_id2,␣
↪date_time_acquisition_start, date_time_acquisition_end],
                         index=['coordinates_id1', 'coordinates_id2',␣
↪'date_time_acquisition_start', 'date_time_acquisition_end'])

    if len(parts) == 10:
        coordinates_id1 = clean_coordinate(parts[3])
        coordinates_id2 = clean_coordinate(parts[4])
        date_time_acquisition_start = parts[7].replace('T', '')
        date_time_acquisition_end = parts[8].replace('T', '')
        return pd.Series([coordinates_id1, coordinates_id2,␣
↪date_time_acquisition_start, date_time_acquisition_end],
                         index=['coordinates_id1', 'coordinates_id2',␣
↪'date_time_acquisition_start', 'date_time_acquisition_end'])

    if len(parts) == 9:
        coordinates_id1 = clean_coordinate(parts[2])
        coordinates_id2 = clean_coordinate(parts[3])
        date_time_acquisition_start = parts[6].replace('T', '')
        date_time_acquisition_end = parts[7].replace('T', '')
        return pd.Series([coordinates_id1, coordinates_id2,␣
↪date_time_acquisition_start, date_time_acquisition_end],
                         index=['coordinates_id1', 'coordinates_id2',␣
↪'date_time_acquisition_start', 'date_time_acquisition_end'])

    if len(parts) == 8:
        coordinates_id1 = clean_coordinate(parts[1])
        coordinates_id2 = clean_coordinate(parts[2])
        date_time_acquisition_start = parts[5].replace('T', '')
        date_time_acquisition_end = parts[6].replace('T', '')
        return pd.Series([coordinates_id1, coordinates_id2,␣
↪date_time_acquisition_start, date_time_acquisition_end],
                         index=['coordinates_id1', 'coordinates_id2',␣
↪'date_time_acquisition_start', 'date_time_acquisition_end'])
```

```
    return pd.Series([None]*4, index=['coordinates_id1', 'coordinates_id2',␣
 ↪'date_time_acquisition_start', 'date_time_acquisition_end'])

# Aplicar a função a cada nome de arquivo no dataset
file_features = train_data['file_name_l1'].apply(extract_features_from_filename)
file_features_test = test_data['file_name_l1'].
 ↪apply(extract_features_from_filename)

# Concatenar as novas features ao dataframe original
train_data = pd.concat([train_data, file_features], axis=1)
test_data = pd.concat([test_data, file_features_test], axis=1)
```

[4]:
```
# Remover colunas 'id' e 'file_name_l1'

train_data = train_data.drop(columns=['id', 'file_name_l1'])
test_data = test_data.drop(columns=['file_name_l1'])
```

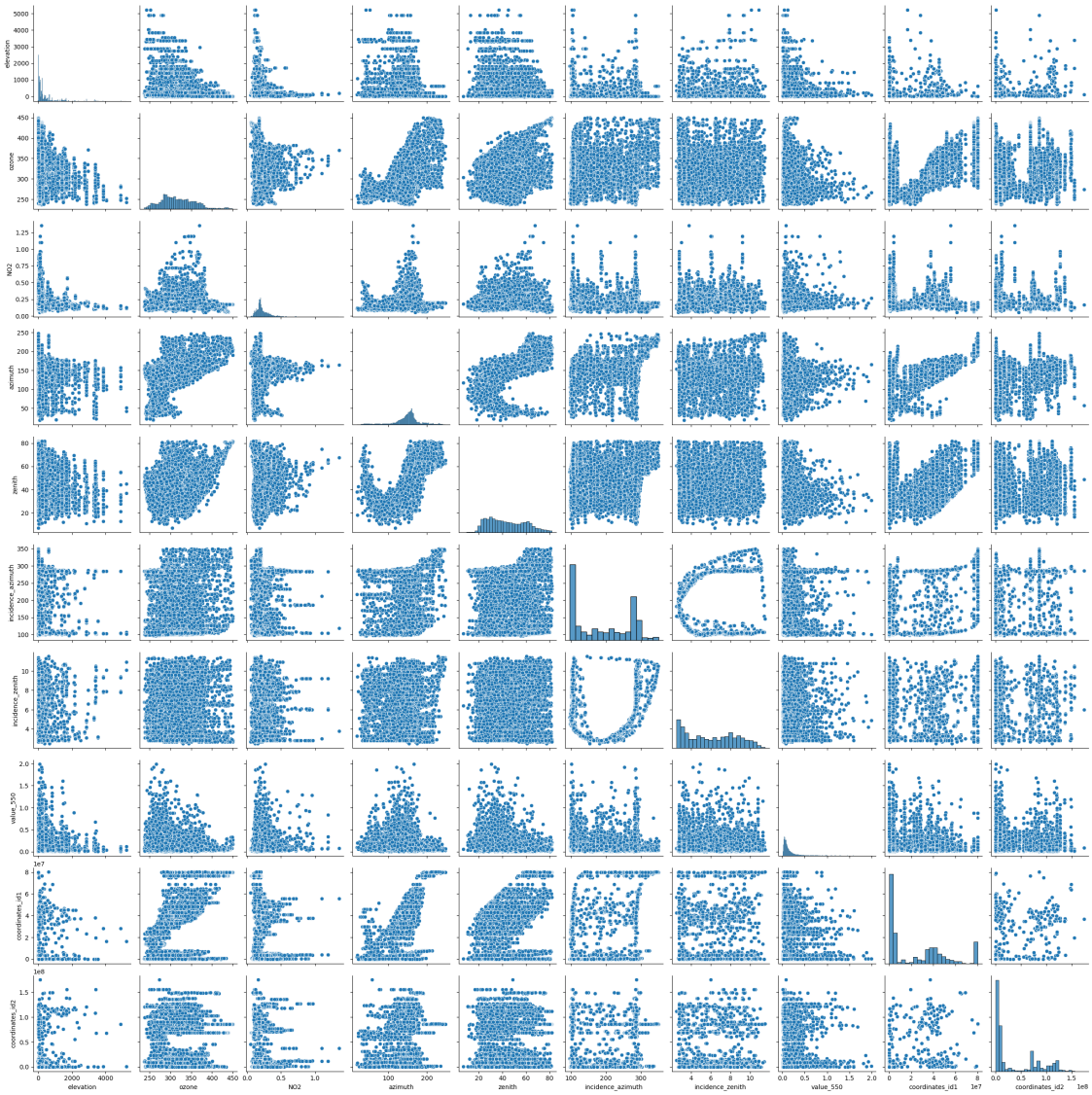
[5]:
```
# Estatísticas descritivas dos dados de treino

print("Estatísticas Descritivas do Dataset de Treino:")
desc_stats = train_data.describe()
print(desc_stats.to_string())
print("\n")

# Verificar valores nulos
print("Verificar Missing Data:")
missing_data = train_data.isnull().sum()
print(missing_data.to_string())
```

```
Estatísticas Descritivas do Dataset de Treino:
          elevation         ozone           NO2       azimuth        zenith
incidence_azimuth  incidence_zenith     value_550  coordinates_id1
coordinates_id2
count  10438.000000  10438.000000  10438.000000  10438.000000  10438.000000
10438.000000      10438.000000  10438.000000     1.043800e+04     1.043800e+04
mean     417.992240    317.283388      0.226330    148.246685     43.553870
191.787363          6.088504      0.143846     2.359460e+07     3.621300e+07
std      671.904367     39.412705      0.101593     33.901711     15.653371
78.862966          2.490754      0.165720     2.582994e+07     4.428080e+07
min      -32.000000    237.000000      0.054000     18.700000      7.000000
96.700000          2.500000      0.010000     1.350000e+02     4.400000e+01
25%       51.000000    288.000000      0.171000    137.100000     30.200000
106.000000          3.600000      0.055000     4.871160e+05     7.813950e+05
50%      174.000000    313.000000      0.201000    153.200000     41.500000
180.700000          6.000000      0.093000     5.859417e+06     1.059290e+07
75%      423.000000    344.000000      0.263000    163.700000     56.700000
283.100000          8.200000      0.166000     4.196685e+07     7.126862e+07
```

```
max     5233.000000    450.000000       1.356000    248.500000     82.000000
349.800000          11.500000      1.985000     8.005361e+07     1.747681e+08


Verificar Missing Data:
elevation                     0
ozone                         0
NO2                           0
azimuth                       0
zenith                        0
incidence_azimuth             0
incidence_zenith              0
value_550                     0
coordinates_id1               0
coordinates_id2               0
date_time_acquisition_start   0
date_time_acquisition_end     0
```

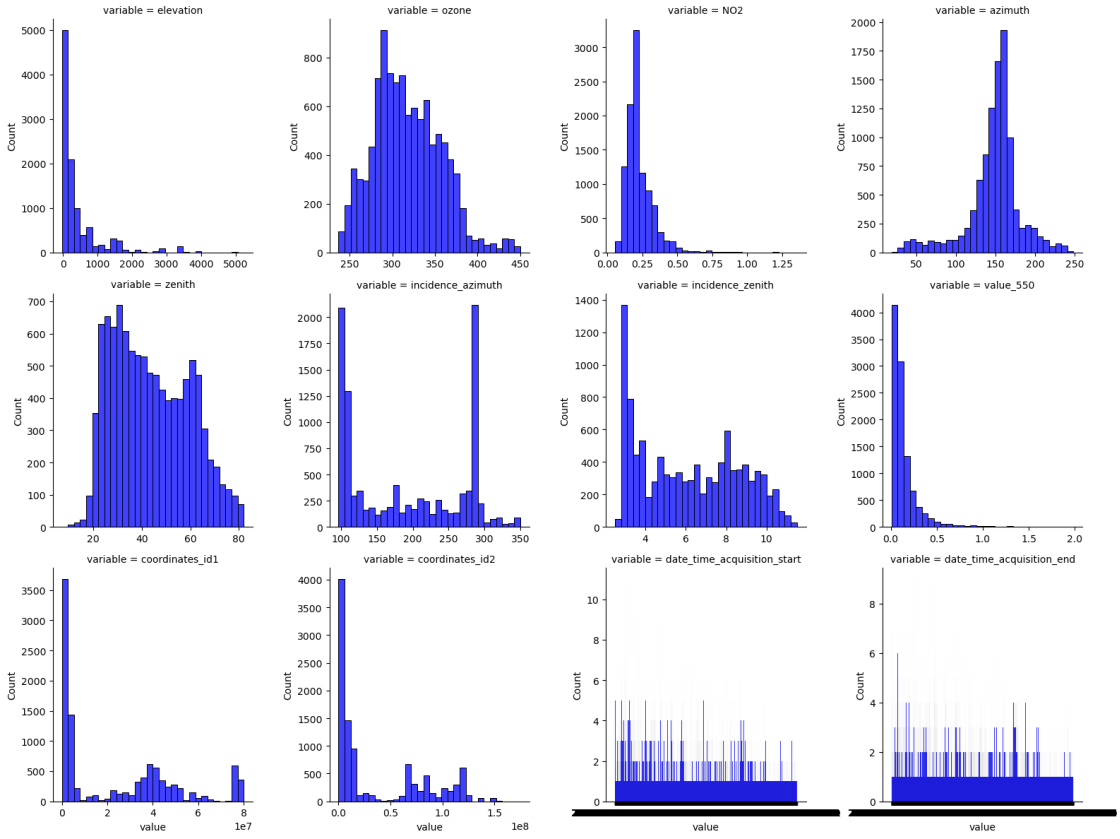
[6]:
```
# Graficos e Visualizações dos dados de treino

numeric_columns = train_data.select_dtypes(include=['number']).columns
sns.pairplot(train_data[numeric_columns])
plt.show()
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



[8]:
```
# Visualizar a distribuição normal das features
df_analise_dist = train_data.melt()

# Histograma para cada feature do DataSet
g = sns.FacetGrid(df_analise_dist, col="variable", col_wrap=4, sharex=False,␣
  ↪sharey=False, height=4)
g.map(sns.histplot, "value", kde=False, color='blue', bins=30)
plt.show()
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



[ ]:
```
3. Análise do Dataset

Pela análise dos dados de treino, podemos observar que:

    Não existem valores nulos no dataset.

    Existem outliers em algumas features, como a elevação e o ozono, mas conclui-se que são valores válidos e não são removidos.

    A distribuição das features é bastante variada, com algumas features a terem uma distribuição normal e outras com uma distribuição mais assimétrica, revelando que a incidencia_azimute e a incidencia_zenite são features que podem não ser as mais relevantes para a predição do valor AOT 550 nm, mas todas as restantes têm alguma corelação com a variável target.
```

[9]:
```
# Valor_550 **COM** ordenação

#train_data_sorted = train_data.sort_values(by='value_550')
#X = train_data_sorted.drop(columns=['value_550'])
```

```
# Valor_550 **SEM** ordenação
#Opção1
X = train_data.drop(columns=['value_550'])

#Opção2
#X = train_data.drop(columns=['value_550','azimuth','zenith'])

#Opção3
#X = train_data_sorted.
 ↪drop(columns=['value_550','azimuth','zenith','date_time_acquisition_start','date_time_acqui



# Target
y = train_data['value_550']



# Dividir os dados de treino completos em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=10)
```

[10]:
```
# Standardizar as features dos dados de treino e validação

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Standardizar as features dos dados completos

scaler = StandardScaler()
X_scaled  = scaler.fit_transform(X)
```

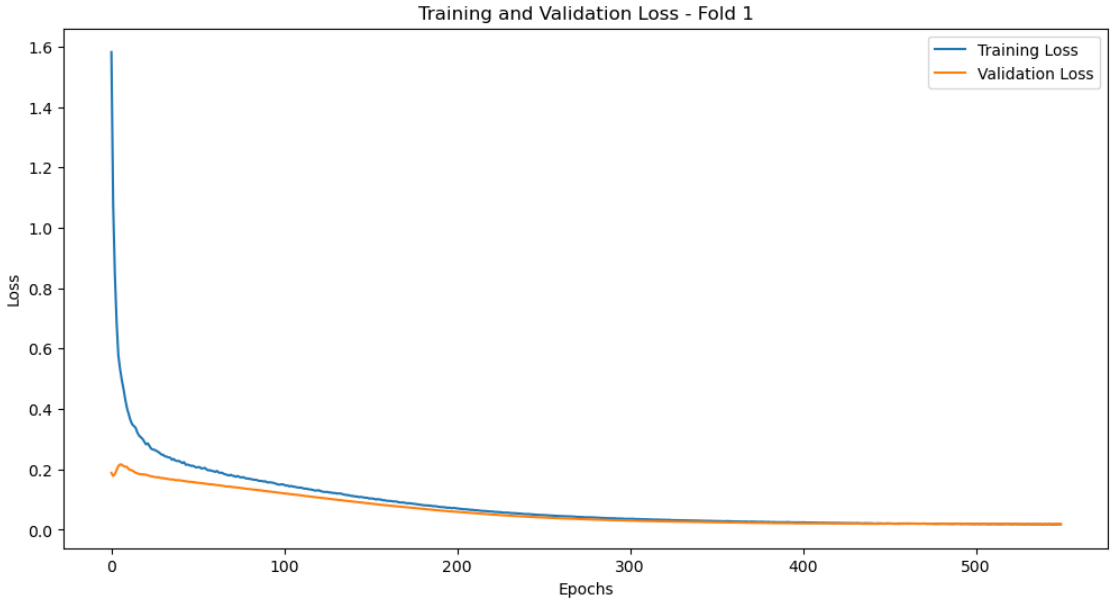
[ ]:
```
4. Design do Setup Exprimental e Construção do Modelo

Os modelos construidos são assim divididos em 2 fases, a primeira com os dados␣
 ↪divididos em treino e validação, e a segunda com os dados de treino␣
 ↪completos aquando a descoberta de uma arqitetura estavel de modo a utilizar␣
 ↪a totalidade dos das de treino disponiveis.

    Fase 1: Construção de um modelo de regressão com recurso à bilioteca␣
 ↪Tesorflow, dividido em folds de validação cruzada para avaliar o MAE (mean␣
 ↪absolute error) do modelo criado, com o objectivo de obter a arquitetura␣
 ↪mais estavel e com o valor de MAE mais baixo possivel.
```

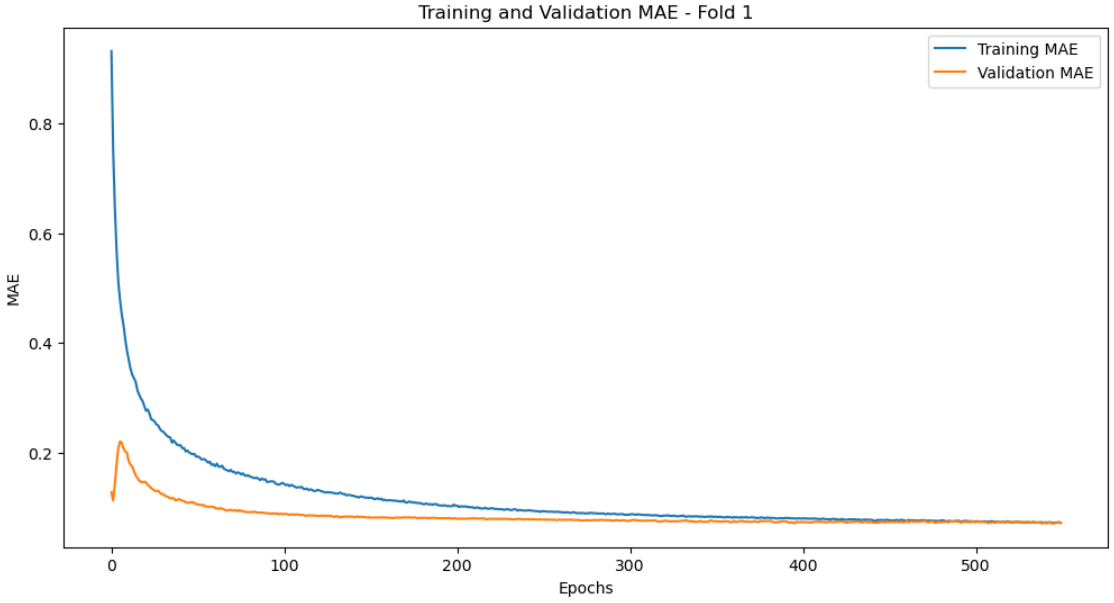
Fase 2: Construção do mesmo modelo utilizado na Fase 1, mas com os dados de treino completos dada a quantidade escassa de dados disponiveis para treino.

[14]:
```
# Criação do modelo com K-Fold Cross Validation

def create_model():
    model = Sequential([
        Dense(200, activation=LeakyReLU(negative_slope=0.01), kernel_regularizer=l1(0.0001),
              kernel_initializer='glorot_uniform', input_shape=(X_train_scaled.shape[1],)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(100, activation=LeakyReLU(negative_slope=0.01), kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.156),
        Dense(50, activation='relu'),
        BatchNormalization(),
        Dropout(0.025),
        Dense(1)
    ])

    optimizer = Adam(learning_rate=0.0003)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])

    return model

# K-Fold Cross Validation
kf = KFold(n_splits=10, shuffle=True, random_state=10)

val_mae_scores = []

# Loop para cada fold
for train_index, val_index in kf.split(X_train_scaled):
    X_train_fold, X_val_fold = X_train_scaled[train_index], X_train_scaled[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    model = create_model()

    # Early stopping callback
    early_stopping = EarlyStopping(monitor='val_loss', patience=25, restore_best_weights=True)
```

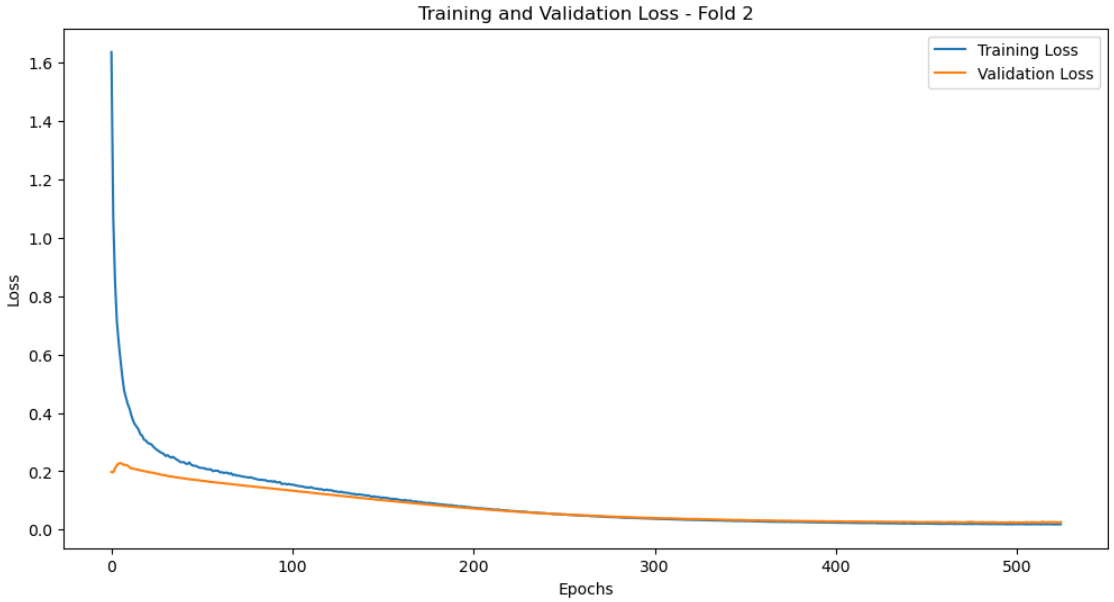
```
    # Treinar o modelo
    history = model.fit(X_train_fold, y_train_fold, epochs=550, batch_size=512,
                        validation_data=(X_val_fold, y_val_fold),␣
 ↪callbacks=[early_stopping], verbose=0)

    # Plotting do training and validation loss para cada fold
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Training and Validation Loss - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Plotting do training and validation MAE para cada fold
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['mae'], label='Training MAE')
    plt.plot(history.history['val_mae'], label='Validation MAE')
    plt.xlabel('Epochs')
    plt.ylabel('MAE')
    plt.title(f'Training and Validation MAE - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Avaliar o modelo no conjunto de validação
    val_loss, val_mae = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    val_mae_scores.append(val_mae)
    print(f"Fold Validation MAE: {val_mae}")

# Calcular a média e desvio padrão dos scores de validação
mean_val_mae = np.mean(val_mae_scores)
std_val_mae = np.std(val_mae_scores)

print(f"\nMean Validation MAE: {mean_val_mae}")
print(f"Standard Deviation of Validation MAE: {std_val_mae}")
```

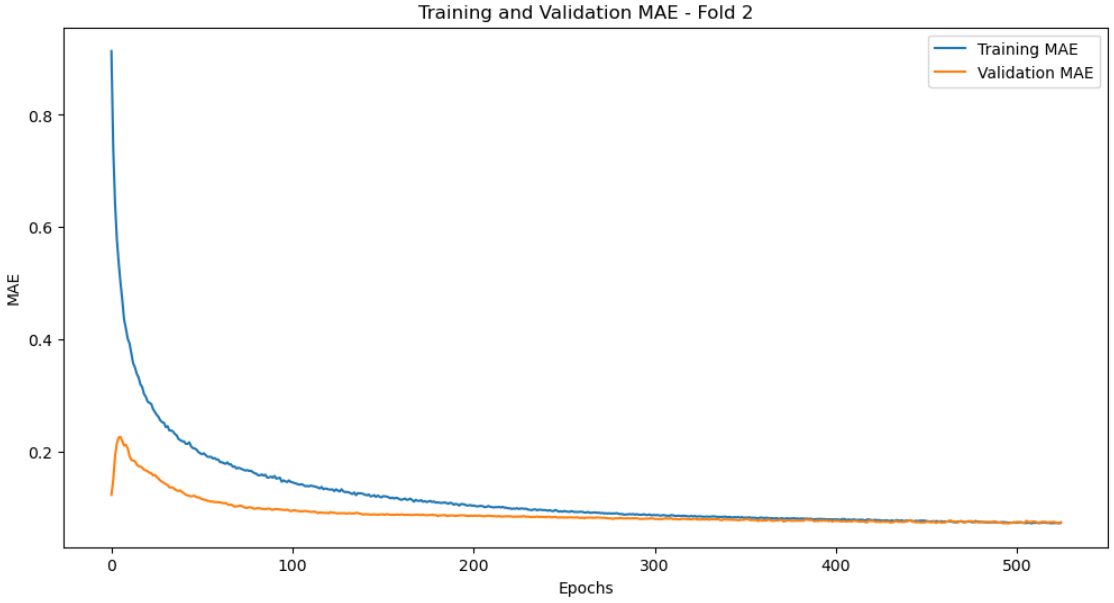
```
C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



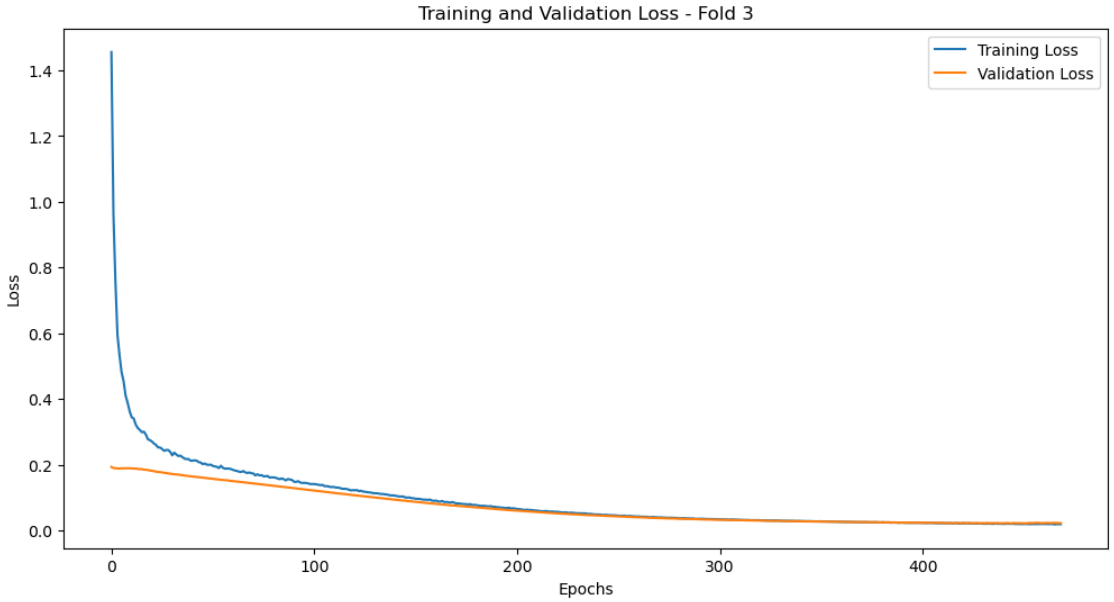


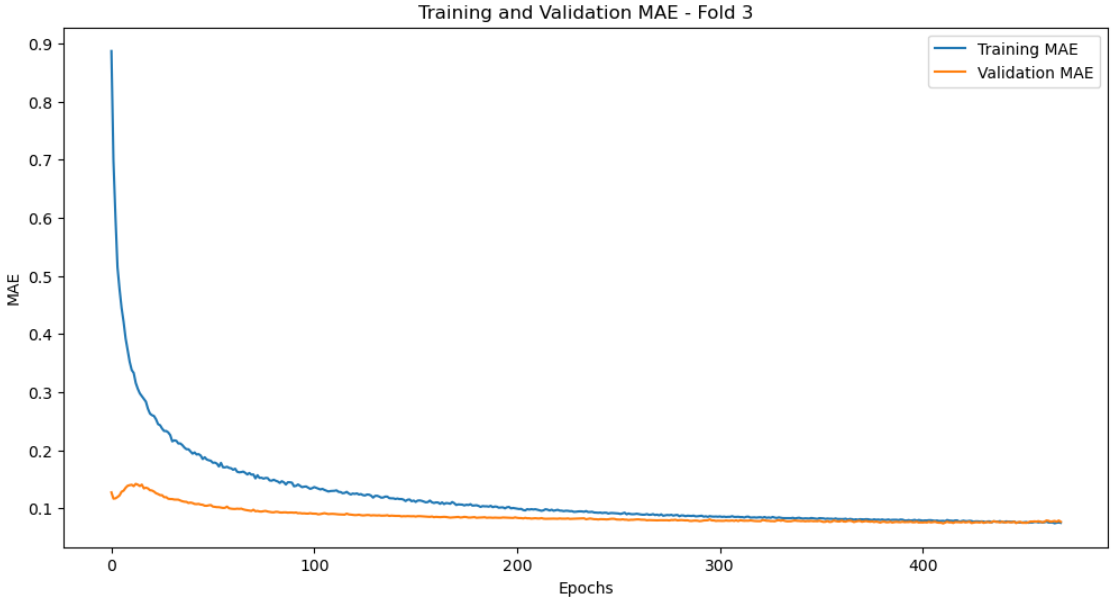

```
Fold Validation MAE: 0.07116591185331345

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



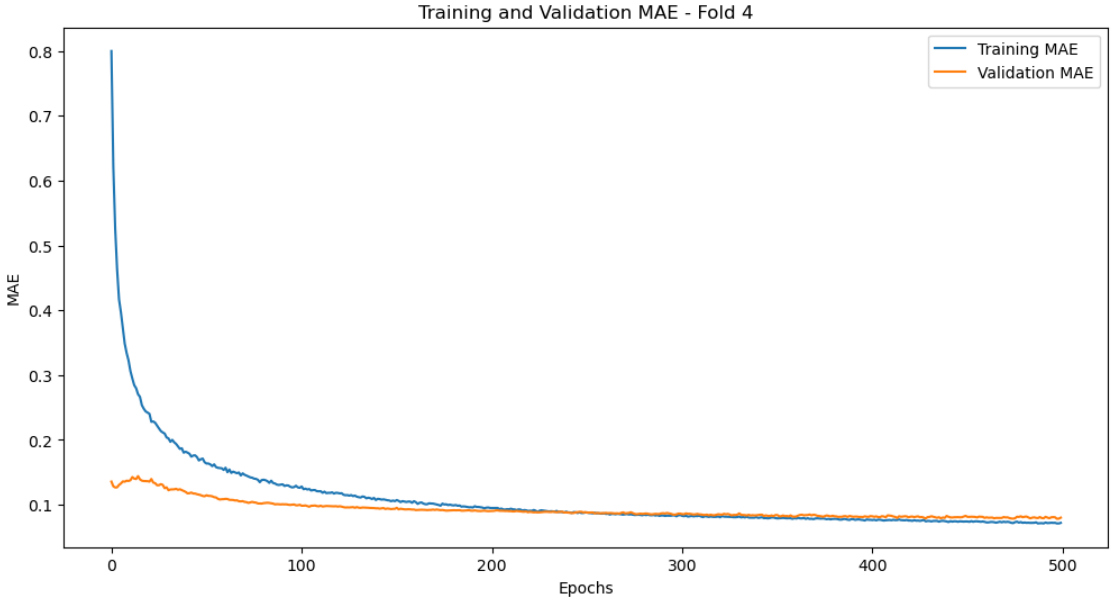



```
Fold Validation MAE: 0.07458700239658356

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

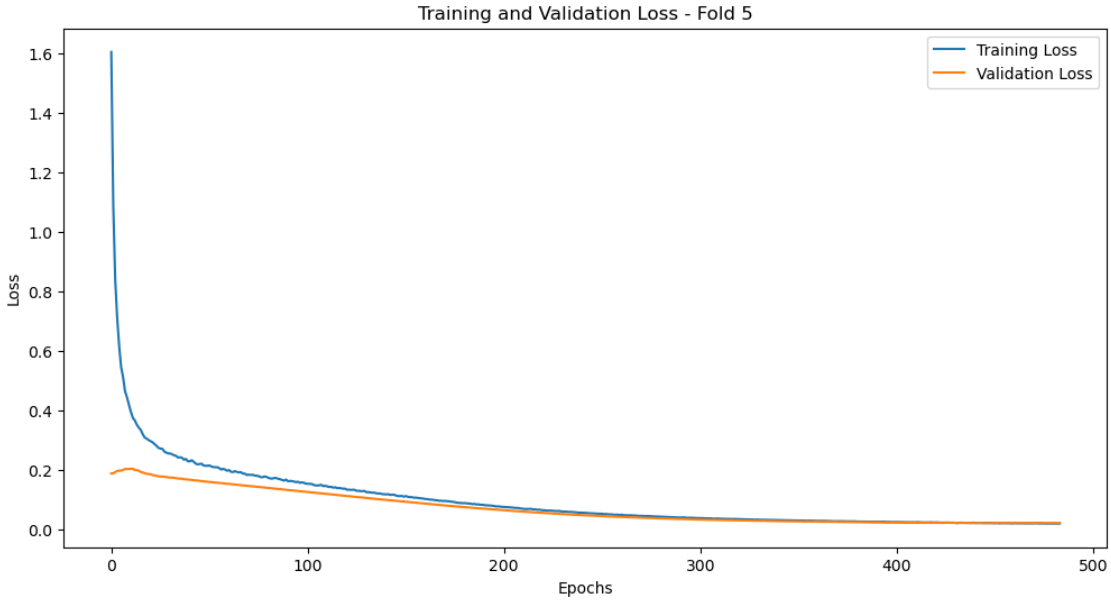


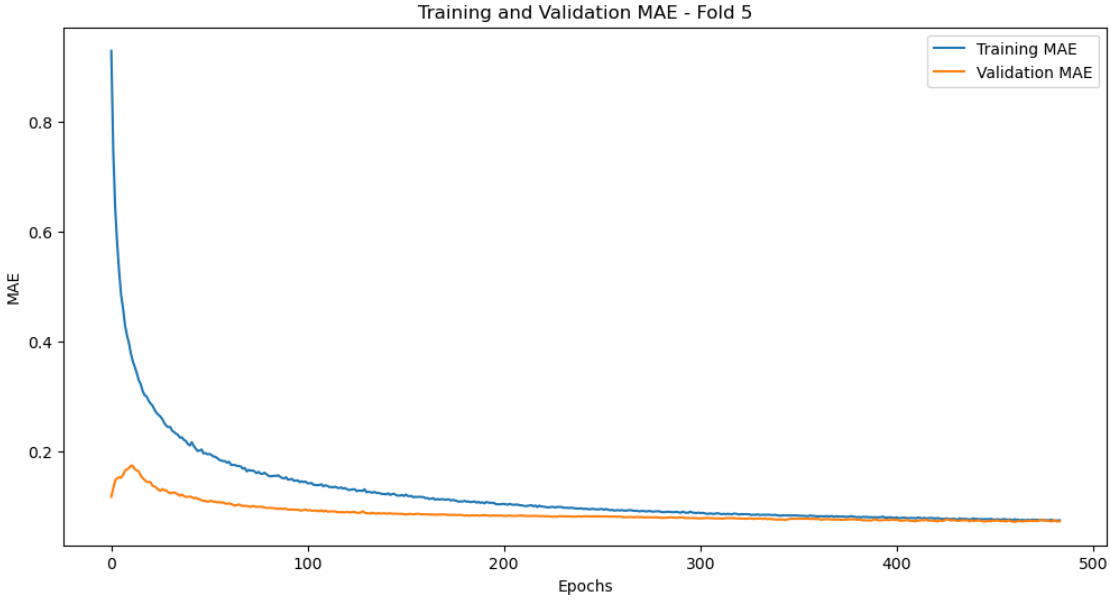



```
Fold Validation MAE: 0.07612760365009308

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

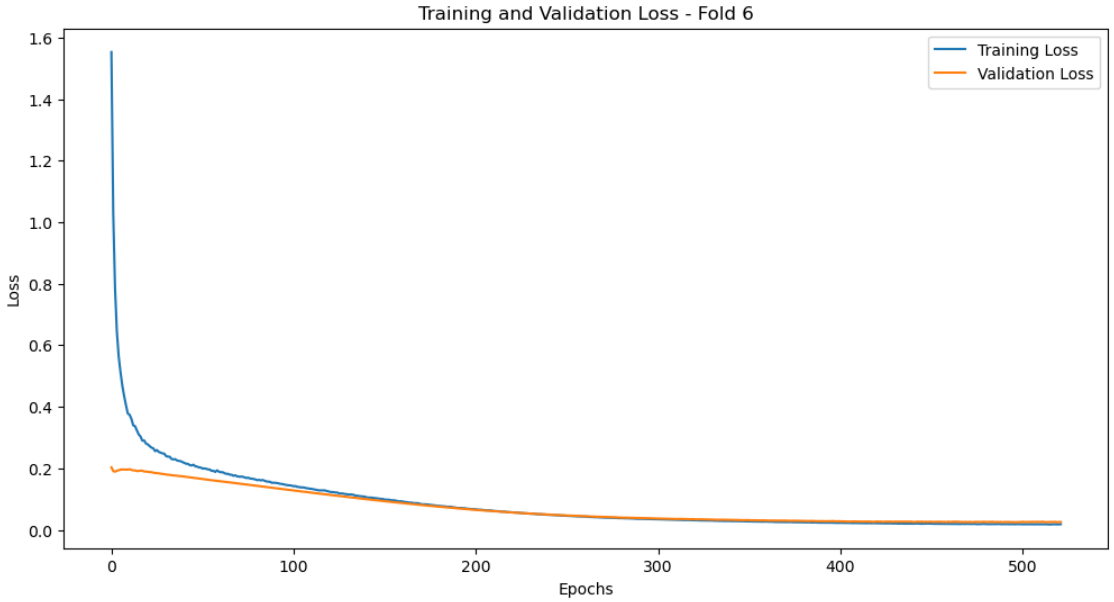


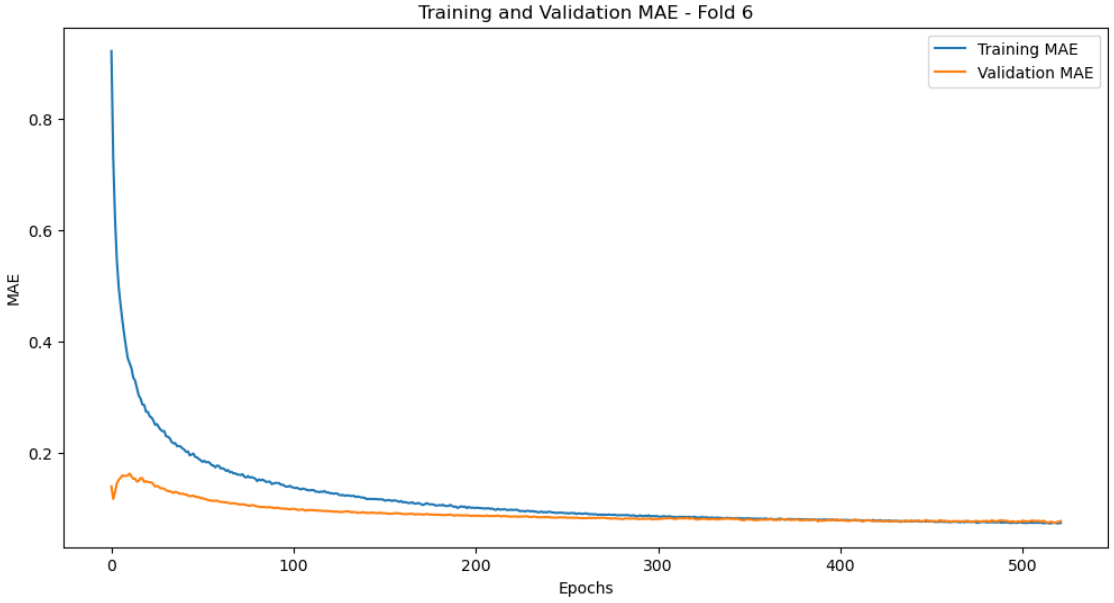

```
Fold Validation MAE: 0.07850497215986252

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



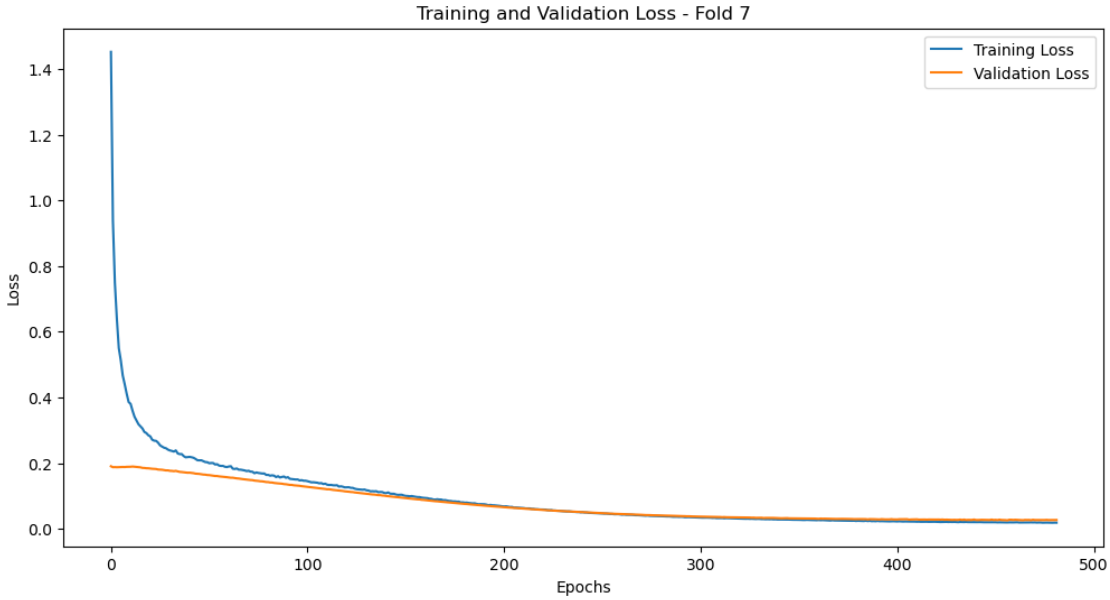


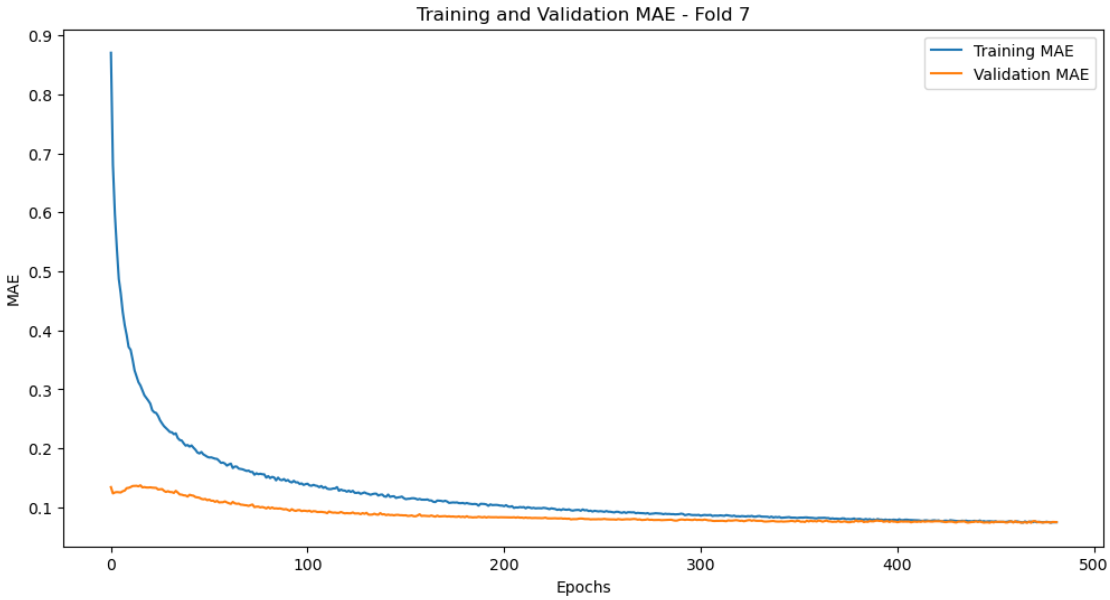

```
Fold Validation MAE: 0.07358454167842865

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



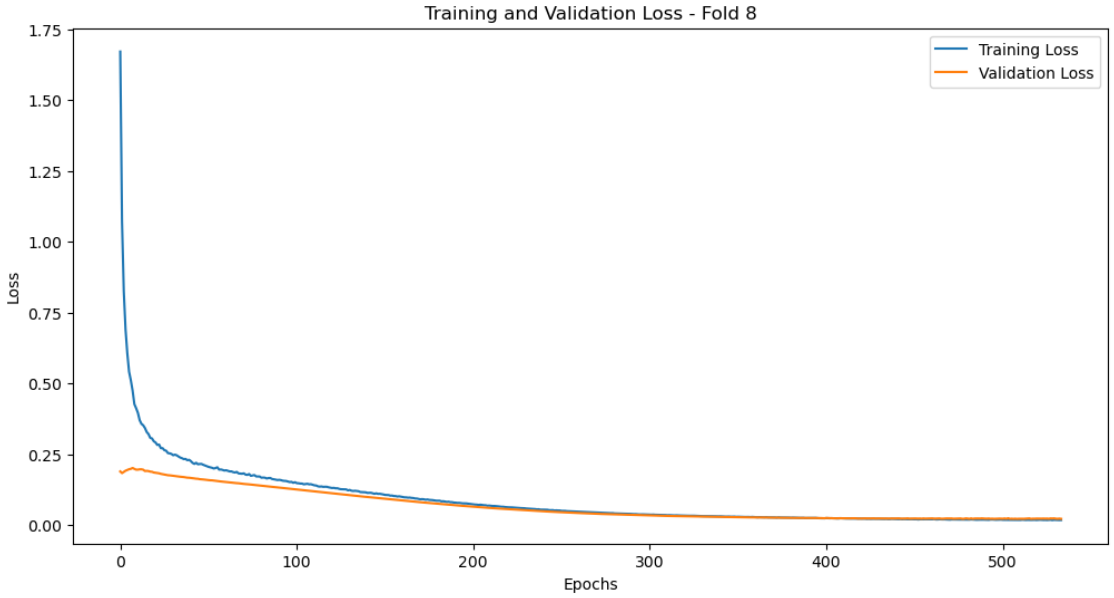


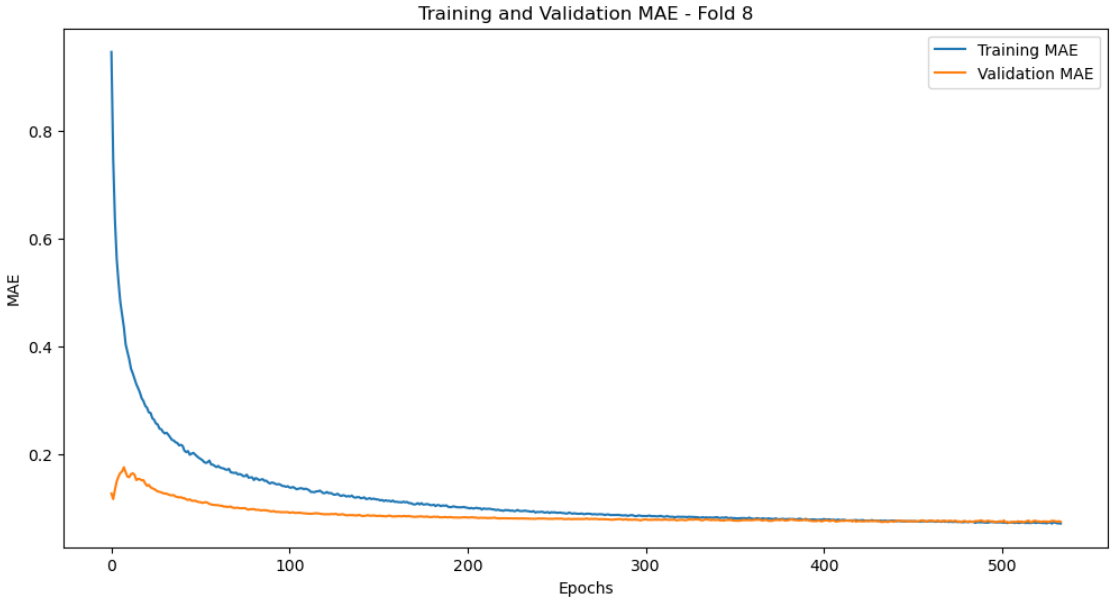

```
Fold Validation MAE: 0.0751485824584961

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



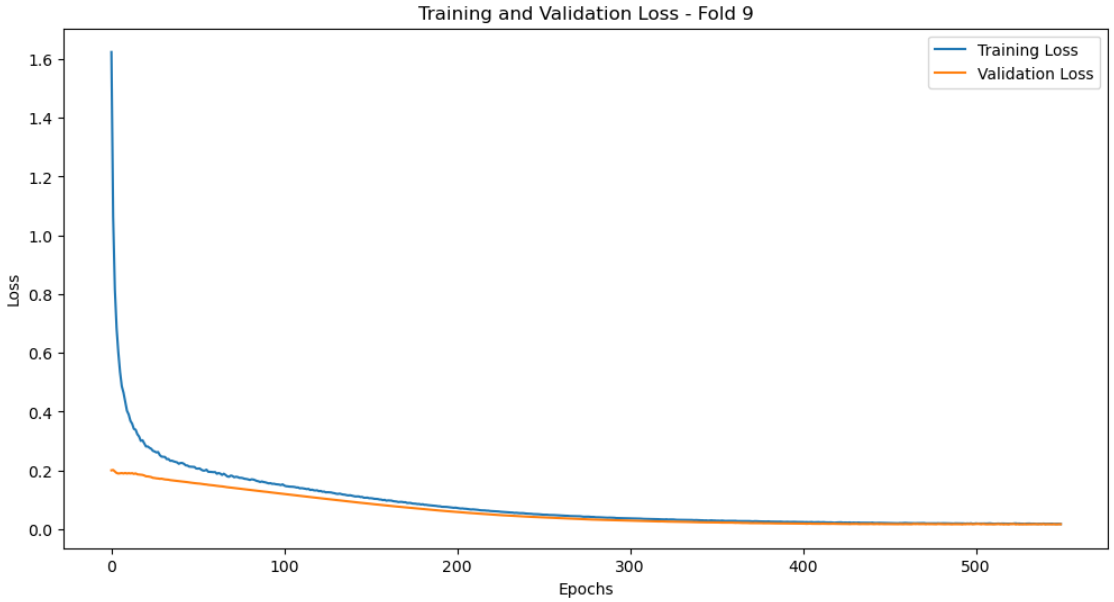


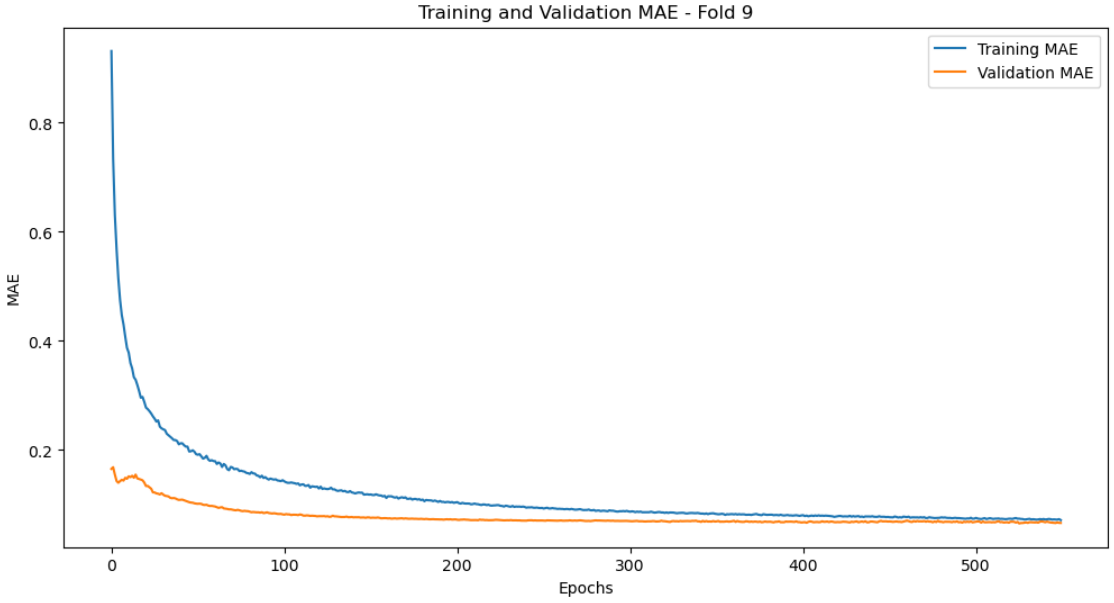

```
Fold Validation MAE: 0.07412929087877274

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



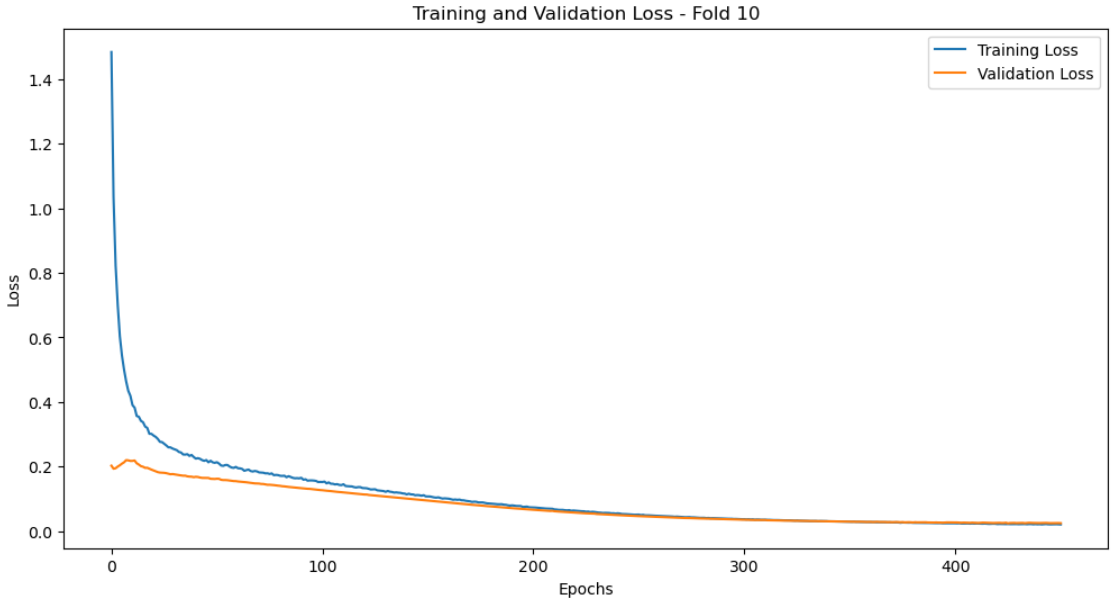


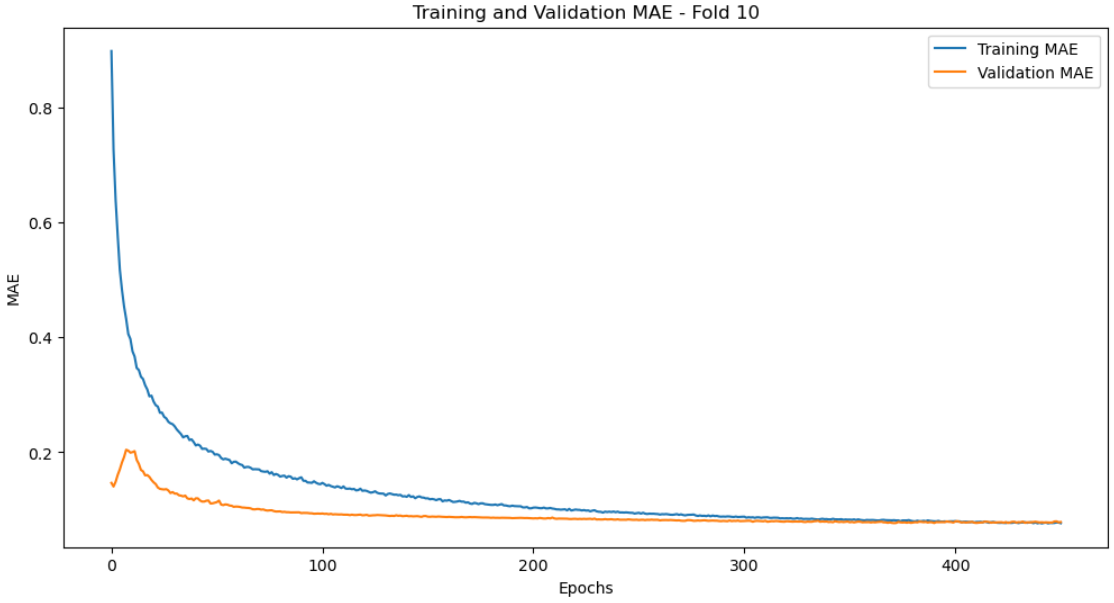

```
Fold Validation MAE: 0.07500236481428146

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```






```
Fold Validation MAE: 0.06533914804458618

C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```






```
Fold Validation MAE: 0.0771249532699585

Mean Validation MAE: 0.07407143712043762
Standard Deviation of Validation MAE: 0.003470389811451106
```

[ ]:
```
4.1 Identificação, descrição e desempenho dos dois melhores modelos
```

Modelo com KFold Cross Validation:

    Houve uma continuação da estrutura simplificada de 3 camadas densas (200, 100, 50) com regularização l1 e l2, e dropout decrescente (0.3, 0.156, 0.025) para prevenir overfitting, intoduzindo a inicialização glorot_uniform para tentar melhorar a estabilidade do modelo.

A arquitetura do modelo nesta iteração é a seguinte:

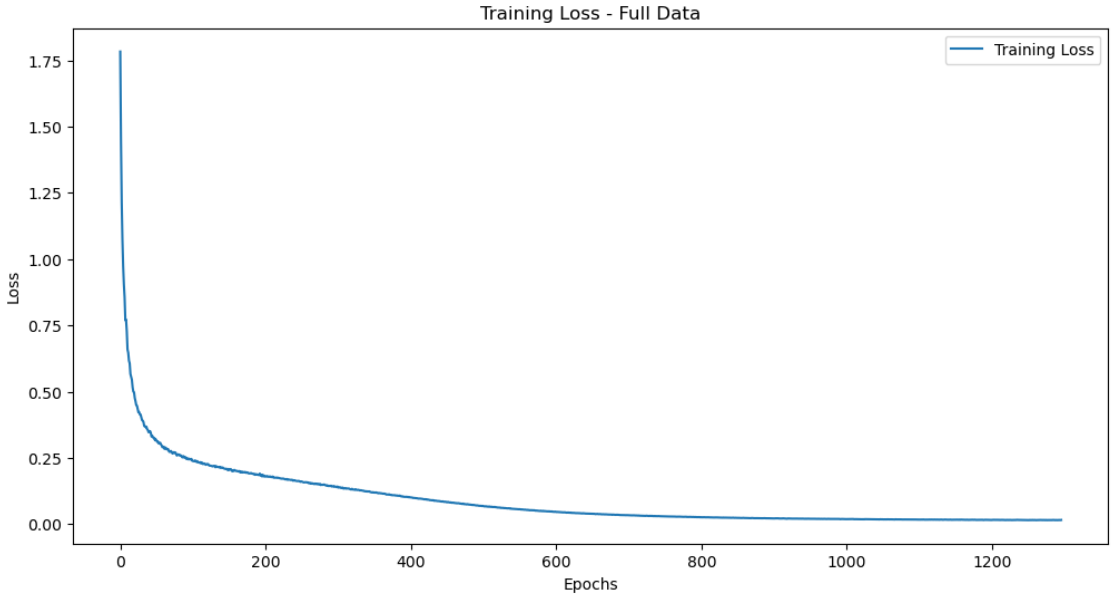
    Camadas: Reduzido para 3 camadas densas (200, 100, 50) para simplificar a arquitetura
    Regularização: l1 e l2 aplicadas de maneira semelhante aos modelos anteriores testando varaiaçoes
    Dropout: Mantido o padrão decrescente (0.3, 0.156, 0.025) para prevenir overfitting
    Ativação: LeakyReLU nas primeiras camadas, ReLU na última camada densa, revelou ser uma boa combinação
    Otimização: Adam com taxa de aprendizagem a 0.0003 e EarlyStopping com paciencia de 25 epochs

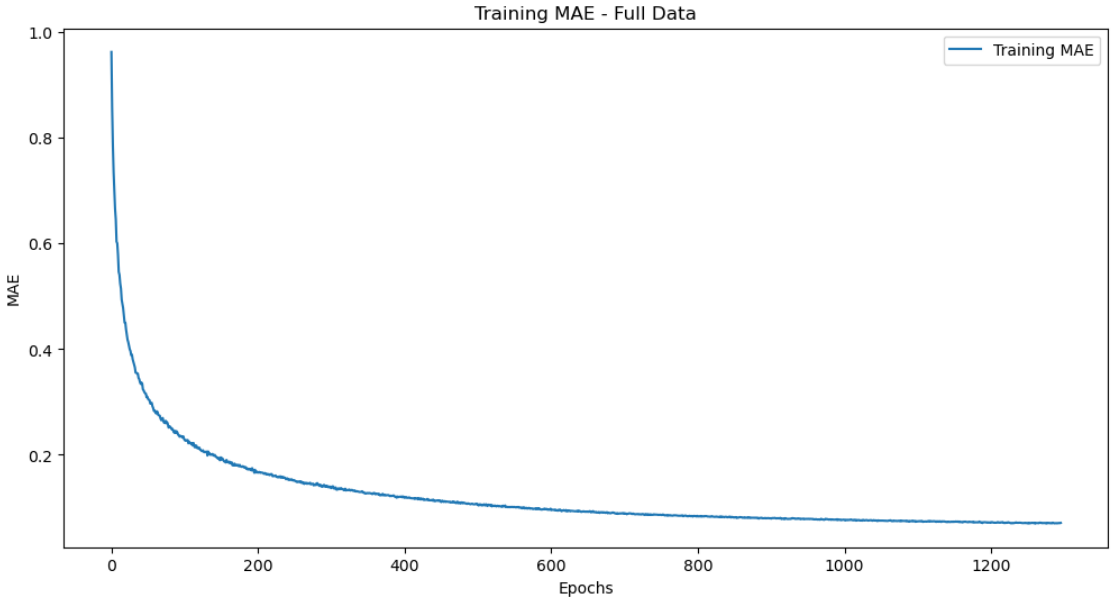
Talvez pela simplicidade da arquitetura, o modelo não conseguiu melhorar o MAE, mas manteve-se estável e com um MAE médio de 0.0739, com um desvio padrão de 0.0035, levando a testes de utilização de diferentes features e diferentes organizações de dados para tentar extrair um melhor MAE, mas com grande dificuldade

[15]:
```python
# Criação do modelo com da dados completos - MAE de 0.0722

def create_model(input_shape):
    model = Sequential([
        Dense(200, activation=LeakyReLU(negative_slope=0.01), kernel_regularizer=l1(0.0001), input_shape=input_shape, kernel_initializer='glorot_uniform'),
        BatchNormalization(),
        Dropout(0.3),
        Dense(100, activation=LeakyReLU(negative_slope=0.01), kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.156),
        Dense(50, activation='relu'),
        BatchNormalization(),
        Dropout(0.025),
        Dense(1)
    ])

    optimizer = Adam(learning_rate=0.0001)
```

```
    model.compile(optimizer=optimizer, loss='mean_squared_error',␣
 ↪metrics=['mae'])

    return model

# Criar o modelo final
final_model = create_model((X_scaled.shape[1],))

# Early stopping callback
early_stopping = EarlyStopping(monitor='loss', patience=35,␣
 ↪restore_best_weights=True)

# Treinar o modelo final em todos os dados disponíveis
history = final_model.fit(X_scaled, y, epochs=2000, batch_size=1024,␣
 ↪callbacks=[early_stopping], verbose=0)

# Plotting do training loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss - Full Data')
plt.legend()
plt.show()

# Plotting do training MAE
plt.figure(figsize=(12, 6))
plt.plot(history.history['mae'], label='Training MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.title('Training MAE - Full Data')
plt.legend()
plt.show()
```

```
C:\Users\Jojo\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```





[ ]:
```
4.2 Identificação, descrição e desempenho dos dois melhores modelos

Modelo final com dados completos:
```

```
    A arquitetura do modelo final é semelhante ao modelo com KFold Cross␣
 ↪Validation, com a diferença de que foi treinado com todos os dados␣
 ↪disponíveis, de modo a tentar extrair o máximo de informação dos dados para␣
 ↪prever o valor AOT 550 nm

A arquitetura do modelo final é a seguinte:

    Camadas: Reduzido para 3 camadas densas (200, 100, 50) para simplificar a␣
 ↪arquitetura
    Regularização: l1 e l2 aplicadas de maneira semelhante aos modelos␣
 ↪anteriores testando varaiaçoes
    Dropout: Mantido o padrão decrescente (0.3, 0.156, 0.025) para prevenir␣
 ↪overfitting
    Ativação: LeakyReLU nas primeiras camadas, ReLU na última camada densa,␣
 ↪revelou ser uma boa combinação
    Otimização: Adam com taxa de aprendizagem a 0.0001 e EarlyStopping com␣
 ↪paciencia de 35 epochs

Neste modelo final, por já ser possivel utilizar todos os dados disponiveis␣
 ↪para treino, o modelo conseguiu melhorar o MAE para 0.0722, revelando que a␣
 ↪utilização de mais dados para treino é uma mais valia para a construção de␣
 ↪um modelo mais robusto e com melhor capacidade de generalização
Nesse sentido foi também possivel aumentar o batch_size para 1024, de modo a␣
 ↪acelerar o treino do modelo e reduzir o tempo de execução e como tal,␣
 ↪aumentar a paciencia do EarlyStopping para 35 epochs, de modo a garantir que␣
 ↪o modelo não para de treinar antes de atingir o ponto de convergência,
com um total de 2000 epochs, o modelo conseguiu atingir o ponto de convergência␣
 ↪e obter um MAE de 0.0722.
```

[13]:
```python
# Standardizar os dados de teste para fazer previsões
test_data_scaled = scaler.transform(test_data.drop(columns=['id']))
```

[ ]:
```python
# Gerar prediçoes
predictions = model.predict(test_data_scaled)

# Preparar submisao em csv
submission = pd.DataFrame({
    'id': test_data['id'],
    'value_550': predictions.flatten()
})

submission.to_csv('submission.csv', index=False)

print("Ficheiro de submissão criado com sucesso!")
```

[ ]:
```
5. Detalhe dos Modelos Construidos e Problemas Encontrados

Durante a iteração dos modelos, foram encontrados alguns problemas que foram␣
 ↪resolvidos com a adição de regularização L1 e L2, e com a adição de Dropout␣
 ↪e BatchNormalization, de modo a evitar overfitting e a melhorar a␣
 ↪generalização do modelo, ou então
com a adição de mais camadas e neurónios para melhorar a capacidade de␣
 ↪aprendizagem do modelo.

Um dos primeiro problemas foi a errada construção do setup de teste, onde os␣
 ↪dados de treino eram standadizados e alimentados ao modelo, mas os dados de␣
 ↪teste não, o que levava a previsões erradas e a um MAE muito elevado, como␣
 ↪verificado nas primeiras submissões.

Outro problema encontrado foi a escolha da arquitetura do modelo, onde a adição␣
 ↪de mais camadas e neurónios não levava a uma melhoria significativa do␣
 ↪modelo, mas sim a um aumento do tempo de treino e a um aumento do MAE, o que␣
 ↪levou a uma escolha de uma arquitetura mais simples e com menos camadas e␣
 ↪neurónios, mas com regularização e Dropout, de modo a melhorar a␣
 ↪generalização do modelo.

Por fim, a escolha do learning rate foi um dos problemas encontrados, onde um␣
 ↪learning rate muito elevado levava a um modelo instavel que saltava a função␣
 ↪de erro e não estabilizava no minimo verdadeiro, mas em minimos locais, por␣
 ↪oposição um learning rate muito baixo levava a um modelo que não aprendia e␣
 ↪que não melhorava o MAE, o que levou a escolha de um learning rate␣
 ↪intermédio que permitisse a convergencia do modelo e a estabilização do MAE.
```

[ ]:
```
# Enumeração dos modelos submetidos na plataforma e seu desempenho

# Modelo Incial - MAE de 0.0850

def create_model():
    model = Sequential([
        Dense(1280, activation='relu', kernel_regularizer=l2(0.0001),␣
 ↪input_shape=(X_train_scaled.shape[1],)),
        BatchNormalization(),
        Dropout(0.5),
        Dense(640, activation='relu', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.5),
        Dense(320, activation='relu', kernel_regularizer=l2(0.01)),
        BatchNormalization(),
        Dropout(0.2),
        Dense(1)  # Output layer for regression
    ])
```

```
    # Compile the model with a lower learning rate
    optimizer = Adam(learning_rate=0.0001)
    model.compile(optimizer=optimizer, loss='mean_squared_error',␣
 ↪metrics=['mae'])

    return model


# Define the KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Store validation results
val_mae_scores = []

for train_index, val_index in kf.split(X_scaled):
    X_train, X_val = X_scaled[train_index], X_scaled[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Create a new model instance
    model = create_model()

    # Early stopping callback
    early_stopping = EarlyStopping(monitor='val_loss', patience=20,␣
 ↪restore_best_weights=True)

    # Train the model
    history = model.fit(X_train, y_train, epochs=300, batch_size=32,␣
 ↪validation_data=(X_val, y_val), callbacks=[early_stopping], verbose=1
                        )

    # Print model summary
    model.summary()

    # Evaluate the model on the validation set
    val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
    val_mae_scores.append(val_mae)
    print(f"Fold Validation MAE: {val_mae}")

# Calculate the mean and standard deviation of the validation MAE scores
mean_val_mae = np.mean(val_mae_scores)
std_val_mae = np.std(val_mae_scores)

print(f"\nMean Validation MAE: {mean_val_mae}")
print(f"Standard Deviation of Validation MAE: {std_val_mae}")
```

[ ]:
```
6. Enumeração dos modelos submetidos em kaggle e o seu desempenho
```

```
Os modelos iniciais eram caracterizados por uma arquitetura densa com várias␣
 ↪camadas e neurónios, com regularização l2 e dropout, mas com uma taxa de␣
 ↪aprendizado muito baixa, o que levava a uma convergência lenta e a um MAE␣
 ↪elevado

A arquitetura do modelo inicial era a seguinte:

    Camadas: 3 camadas densas com unidades decrescentes (1280, 640, 320)
    Regularização: l2 com valores variáveis (0.0001, 0.001, 0.01)
    Dropout: Alta taxa de dropout (0.5, 0.5, 0.2) para prevenir overfitting
    Ativação: ReLU para todas as camadas
    Otimização: Adam com taxa de aprendizagem de 0.0001

A estrutura inicial é bastante densa com uma taxa de dropout demasiado elevada␣
 ↪que não capturava correctamente padrões nos dados. A regularização l2 está␣
 ↪bem distribuída, mas a taxa de aprendizado é baixa, o que leva a uma␣
 ↪convergência lenta, associado a um batch_zize muito pequeno que leva a um␣
 ↪tempo para treino muito elevado.
```

[ ]:
```
# Enumeração dos modelos submetidos na plataforma e seu desempenho

# Modelo Intermedio -  MAE de 0.0790

def create_model():
    model = Sequential([
        Dense(200, activation=LeakyReLU(negative_slope=0.01),␣
 ↪kernel_regularizer=l1(0.0001), input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(100, activation=LeakyReLU(negative_slope=0.01),␣
 ↪kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.156),
        Dense(100, activation=LeakyReLU(negative_slope=0.01),␣
 ↪kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.156),
        Dense(100, activation=LeakyReLU(negative_slope=0.01),␣
 ↪kernel_regularizer=l2(0.002)),
        BatchNormalization(),
        Dropout(0.125),
        Dense(100, activation=LeakyReLU(negative_slope=0.01),␣
 ↪kernel_regularizer=l2(0.002)),
        BatchNormalization(),
        Dropout(0.072),
        Dense(50, activation='relu'),
        BatchNormalization(),
```

```
        Dropout(0.025),
        Dense(1)
    ])


    optimizer = Adam(learning_rate=0.0004)
    model.compile(optimizer=optimizer, loss='mean_squared_error',␣
 ↪metrics=['mae'])

    return model

# K-Fold Cross Validation
kf = KFold(n_splits=10, shuffle=True, random_state=10)


val_mae_scores = []

# Loop para cada fold
for train_index, val_index in kf.split(X_train_scaled):
    X_train_fold, X_val_fold = X_train_scaled[train_index],␣
 ↪X_train_scaled[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
 ↪iloc[val_index]


    model = create_model()

    # Early stopping callback
    early_stopping = EarlyStopping(monitor='val_loss', patience=25,␣
 ↪restore_best_weights=True)

    # Trainar o modelo
    history = model.fit(X_train_fold, y_train_fold, epochs=550, batch_size=512,␣
 ↪validation_data=(X_val_fold, y_val_fold), callbacks=[early_stopping],␣
 ↪verbose=1)

    # Plotting do training and validation loss para cada fold
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Training and Validation Loss - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Plotting do training and validation MAE para cada fold
```

```
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['mae'], label='Training MAE')
    plt.plot(history.history['val_mae'], label='Validation MAE')
    plt.xlabel('Epochs')
    plt.ylabel('MAE')
    plt.title(f'Training and Validation MAE - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Avaliar o modelo no conjunto de validação
    val_loss, val_mae = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    val_mae_scores.append(val_mae)
    print(f"Fold Validation MAE: {val_mae}")

# Calcular a média e desvio padrão dos scores de validação
mean_val_mae = np.mean(val_mae_scores)
std_val_mae = np.std(val_mae_scores)

print(f"\nMean Validation MAE: {mean_val_mae}")
print(f"Standard Deviation of Validation MAE: {std_val_mae}")
```

[ ]:
```
6. Enumeração dos modelos submetidos em kaggle e o seu desempenho

Na segunda iteração de modelos, a arquitetura foi complicada com mais camadas e␣
 ↪neurónios numa tentativa de capturar melhor os padroes existentes, mas com␣
 ↪sucesso reduzido, com regularização L1 e L2, e com uma taxa de aprendizado␣
 ↪mais elevada, o que levou a uma convergência mais rápida e a uma␣
 ↪estabilização do MAE

As arquiteturas do modelo nesta iteração era a seguinte:

    Camadas: 6 camadas densas com unidades decrescentes (200, 100, 100, 100,␣
 ↪100, 50) de modo a capturar padrões mais complexos
    Regularização: Combinação de l1 e l2 (0.0001 e 0.002, respectivamente)
    Dropout: Dropout moderado decrescente (0.3, 0.156, 0.125, 0.072, 0.025)
    Ativação: LeakyReLU para as primeiras camadas, ReLU para a penúltima camada
    Otimização: Adam com taxa de aprendizagem de 0.0004

Nesta iteração a estrutura é mais profunda e equilibrada, com regularização l1␣
 ↪no início para controle de disperção dos dados e l2 para penalizar grandes␣
 ↪pesos nas camadas mais acima. A taxa de dropout é ajustada para cada camada,␣
 ↪o que pode ajudar a regularização sem perder muita informação,
mas a grande alteração foi na taxa de batch_size que foi aumentada para 512, o␣
 ↪que levou a uma convergência mais rápida e a uma estabilização do MAE, sendo␣
 ↪tentando varios valores até encontrar o mais estavel onde nao houvesse␣
 ↪comportamento erratico na maquina.
```

```
[ ]: # Enumeração dos modelos submetidos na plataforma e seu desempenho

     # Modelo Intermedio - MAE de 0.0750

     def create_model():
         model = Sequential([
             Dense(200, activation=LeakyReLU(negative_slope=0.01),␣
      ↪kernel_regularizer=l1(0.0001), input_shape=(X_train.shape[1],)),
             BatchNormalization(),
             Dropout(0.3),
             Dense(100, activation=LeakyReLU(negative_slope=0.01),␣
      ↪kernel_regularizer=l2(0.001)),
             BatchNormalization(),
             Dropout(0.156),
             Dense(50, activation='relu'),
             BatchNormalization(),
             Dropout(0.025),
             Dense(1)
         ])


         optimizer = Adam(learning_rate=0.0003)
         model.compile(optimizer=optimizer, loss='mean_squared_error',␣
      ↪metrics=['mae'])

         return model

     # K-Fold Cross Validation
     kf = KFold(n_splits=10, shuffle=True, random_state=10)


     val_mae_scores = []

     # Loop para cada fold
     for train_index, val_index in kf.split(X_train_scaled):
         X_train_fold, X_val_fold = X_train_scaled[train_index],␣
      ↪X_train_scaled[val_index]
         y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
      ↪iloc[val_index]


         model = create_model()

         # Early stopping callback
         early_stopping = EarlyStopping(monitor='val_loss', patience=25,␣
      ↪restore_best_weights=True)
```

```
    # Trainar o modelo
    history = model.fit(X_train_fold, y_train_fold, epochs=550, batch_size=512,␣
 ↪validation_data=(X_val_fold, y_val_fold), callbacks=[early_stopping],␣
 ↪verbose=1)

    # Plotting do training and validation loss para cada fold
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Training and Validation Loss - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Plotting do training and validation MAE para cada fold
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['mae'], label='Training MAE')
    plt.plot(history.history['val_mae'], label='Validation MAE')
    plt.xlabel('Epochs')
    plt.ylabel('MAE')
    plt.title(f'Training and Validation MAE - Fold {len(val_mae_scores) + 1}')
    plt.legend()
    plt.show()

    # Avaliar o modelo no conjunto de validação
    val_loss, val_mae = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    val_mae_scores.append(val_mae)
    print(f"Fold Validation MAE: {val_mae}")

# Calcular a média e desvio padrão dos scores de validação
mean_val_mae = np.mean(val_mae_scores)
std_val_mae = np.std(val_mae_scores)

print(f"\nMean Validation MAE: {mean_val_mae}")
print(f"Standard Deviation of Validation MAE: {std_val_mae}")
```

[ ]:
```
6. Enumeração dos modelos submetidos em kaggle e o seu desempenho

Na proxima iteração de modelos, a arquitetura foi simplificada com menos␣
 ↪camadas e neurónios, com regularização L1 e L2, e com uma taxa de␣
 ↪aprendizado mais elevada, o que levou a uma convergência mais rápida e a uma␣
 ↪estabilização do MAE, tendo sido testado diferentes valores de kfold que␣
 ↪poderiam afetar a distribuição dos dados,
ordenação dos dados à entrada do modelo, e a remoção de features que não tinham␣
 ↪impacto na predição do modelo, não tendo havido grandes melhorias␣
 ↪significativas.
```

```
As arquiteturas do modelo nesta iteração era a seguinte:

    Camadas: Reduzido para 3 camadas densas (200, 100, 50) para simplificar a␣
 ↪arquitetura
    Regularização: l1 e l2 aplicadas de maneira semelhante aos modelos␣
 ↪anteriores testando varaiaçoes
    Dropout: Mantido o padrão decrescente (0.3, 0.156, 0.025) para prevenir␣
 ↪overfitting
    Ativação: LeakyReLU nas primeiras camadas, ReLU na última camada densa,␣
 ↪revelou ser uma boa combinação
    Otimização: Adam com taxa de aprendizagem a 0.0003 e EarlyStopping com␣
 ↪paciencia de 25 epochs


Esta iteração adota uma estrutura mais simples e eficaz, onde o objectivo␣
 ↪passou pela profundidade do modelo, isto é, reduzi ao maximo o validation␣
 ↪error exsitente para cada um dos 10 folds. A regularização e dropout foram␣
 ↪mantidos para prevenir overfitting e a taxa de aprendizagem mais baixa␣
 ↪garante que o modelo converge de forma estável.
O valor de epochs foi mantido nos 550 devido à restrição do EarlyStopping, que␣
 ↪garante que o modelo não entra em overfitting e não se torna instavel,␣
 ↪divergindo para valores de erro elevados, percebendo  que com o␣
 ↪EarlyStopping o modelo não ultrapassava os 350 a 500 epochs.
```

[ ]:
```
7. Conclusão

No final o nosso melhor modelo atigiu um MAE de 0.0722, com uma arquitetura de␣
 ↪3 camadas densas (200, 100, 50) com regularização l1 e l2, e dropout␣
 ↪decrescente (0.3, 0.156, 0.025) para prevenir overfitting, intoduzindo a␣
 ↪inicialização glorot_uniform para tentar melhorar a estabilidade do modelo,
mas não fuicou nem perto do melhor modelo submetido na plataforma, o que revela␣
 ↪que a arquitetura do modelo pode ter sido um grande erro, ou a optimização␣
 ↪dos hyperparametros uma grande falha.

Tentamos sempre um modelo que fosse o mais profundo possivel, com o valor de␣
 ↪val_error mais baixo possivel, e ao mesmo tempo melhorar o MAE para o mais␣
 ↪baixo possivel, mas sem sucesso. Utilizamos a ligação à maquina Niiaa␣
 ↪atravez de umaponte ssh com um servidor localhost de jupyter notebook, mas␣
 ↪as melhorias foram apenas incrementais para o modelo que já tinhamos␣
 ↪desenvolvido.
```

```
No final, para alem da arquitetura desenvolvida o futuro trabalho passaria pela␣
  ↪utilização do keras tuner para optimização dos hyperparametros na maquina␣
  ↪Niiaa, algo que requeria tempo, mas que poderia ser uma mais valia para a␣
  ↪construção de um modelo mais robusto e com melhor capacidade de␣
  ↪generalização, visto que foram submetidos modelos com scores de 0.005 de MAE.
```