In [44]:

```python
# packages used in this tutorial
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [45]:

```python
# Load the CSV files into dataframes
dataframes = {}
keys = [str(i).zfill(2) for i in range(1, 13)] # strings '01' to '12'
for key in keys:
    df = pd.read_csv(f'CSVafterClean/{key}.csv')
    dataframes[key] = df
```

In [43]:

```python
# Assuming 'dataframes' is your dictionary of dataframes
# Extract the 'prcp_total' column from each dataframe
X = []  # Input features
y = []  # Target variable

for key, df in dataframes.items():
     # Select all columns except 'lat', 'lon', 'time', and 'prcp_total', this mvp neural ne
    features = df.loc[:, ~df.columns.isin(['lat', 'lon', 'time', 'prcp_total'])].values
    X.append(features) #a list of arrays, where each array represents the features for one
    y.append(df['prcp_total'].values) #a list of 1D NumPy arrays, where each array represen

# Combine data from all dataframes
X = np.vstack(X) #vertically stacks (concatenates) these arrays on top of each other, effec
#where each row represents a data point (sample), and each column represents a feature.
y = np.concatenate(y) # y becomes a 1D array of target data point values of the one 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)

# Standardize the input features (optional but often recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build your neural network model
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer with a single neuron for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss = model.evaluate(X_test, y_test)
print(f"Mean Squared Error on Test Set: {loss}")
```

```
Epoch 1/10
2546/2546 [==============================] - 3s 898us/step - loss: 20.7679 - val_loss: 17.957
5
Epoch 2/10
2546/2546 [==============================] - 2s 836us/step - loss: 17.0139 - val_loss: 15.992
9
```

```
Epoch 3/10
2546/2546 [==============================] - 2s 867us/step - loss: 15.4077 - val_loss: 14.897
4
Epoch 4/10
2546/2546 [==============================] - 2s 838us/step - loss: 14.2814 - val_loss: 13.865
3
Epoch 5/10
2546/2546 [==============================] - 2s 801us/step - loss: 13.4850 - val_loss: 13.350
1
Epoch 6/10
2546/2546 [==============================] - 2s 827us/step - loss: 12.7514 - val_loss: 12.378
2
Epoch 7/10
2546/2546 [==============================] - 2s 804us/step - loss: 12.1037 - val_loss: 12.119
0
Epoch 8/10
2546/2546 [==============================] - 2s 769us/step - loss: 11.5166 - val_loss: 11.593
1
Epoch 9/10
2546/2546 [==============================] - 2s 804us/step - loss: 11.0828 - val_loss: 11.614
4
Epoch 10/10
2546/2546 [==============================] - 2s 870us/step - loss: 10.6028 - val_loss: 10.780
5
637/637 [==============================] - 0s 600us/step - loss: 10.7805
Mean Squared Error on Test Set: 10.780468940734863
```

In [39]:

```
#Shape Check
print(f"Shape of y: {y.shape}, shape of X: {X.shape}")
num_features = X.shape[1]
print(f"Number of features in X: {num_features}")
num_samples = X.shape[0]
print(f"Number of data points in X: {num_samples}")
```

```
Shape of y: (101835,), shape of X: (101835, 21)
Number of features in X: 21
Number of data points in X: 101835
```

In [30]:

```
X_train
```

Out[30]:

```
array([[ 1.17601825,  1.46330442,  0.7429586 , ...,  0.84517666,
         0.68569515,  0.97780313],
       [ 1.28664622,  0.91368946,  1.53835749, ...,  0.6314709 ,
         0.49434827,  0.68867771],
       [-0.48153499, -0.51376216, -0.70567777, ...,  0.25542257,
         0.20753991,  0.22248553],
       ...,
       [ 0.49112688, -0.02061638,  0.84900938, ..., -4.08400472,
        -4.00849861, -3.96962024],
       [-0.63295261, -1.30878615, -0.25395839, ..., -0.67956864,
        -0.50887745, -0.6147075 ],
       [-2.04781682, -2.36798131, -1.49133754, ...,  1.54144636,
         1.46983182,  1.7540931 ]])
```

In [38]:

```
y_train
```

Out[38]:

```
array([2.74420997e+00, 4.46081124e-03, 1.98121020e-03, ...,
       2.71070460e-01, 1.26143777e+00, 2.39255380e-01])
```

In [19]:

```
# show a summary of the data
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 dense_6 (Dense)             (None, 64)                1408

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 1)                 33

=================================================================
Total params: 3521 (13.75 KB)
Trainable params: 3521 (13.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [20]:

```python
# Display training progress by printing a single dot for each completed epoch
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

# Function to plot how the model is doing during training
# Visualize the model's training progress using the stats stored in the history object.
# We want to use this data to determine how long to train before the model stops making pro
def plot_history(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error [mm]')
    plt.plot(history.epoch, np.array(history.history['loss']),
           label='Train Loss')
    plt.plot(history.epoch, np.array(history.history['val_loss']),
           label = 'Val loss')
    plt.legend()
    #plt.ylim([0, 5])
```

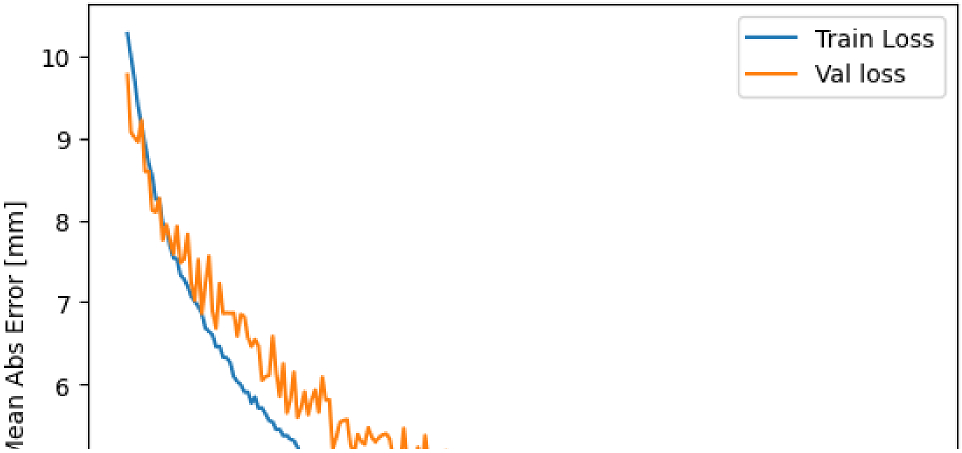
In [21]:

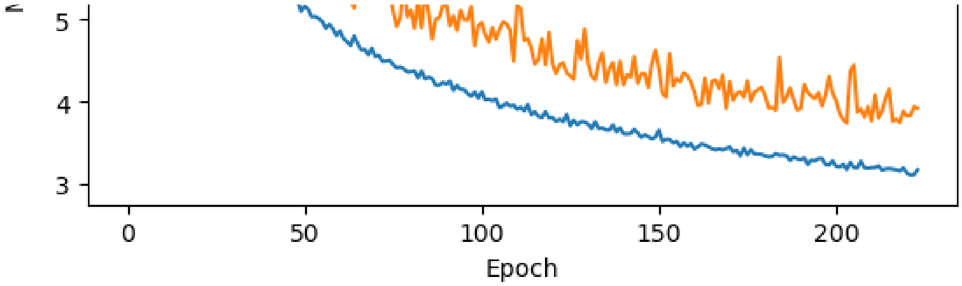
```python
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)

# Store training stats
history = model.fit(X_train, y_train, epochs=1000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history(history)
```

```
.............................................................................................
.......
.............................................................................................
.......
........................
```





In [29]:

```
# Calculate MAE separately
from sklearn.metrics import mean_absolute_error
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error on Test Set: {mae} millimeters")
```

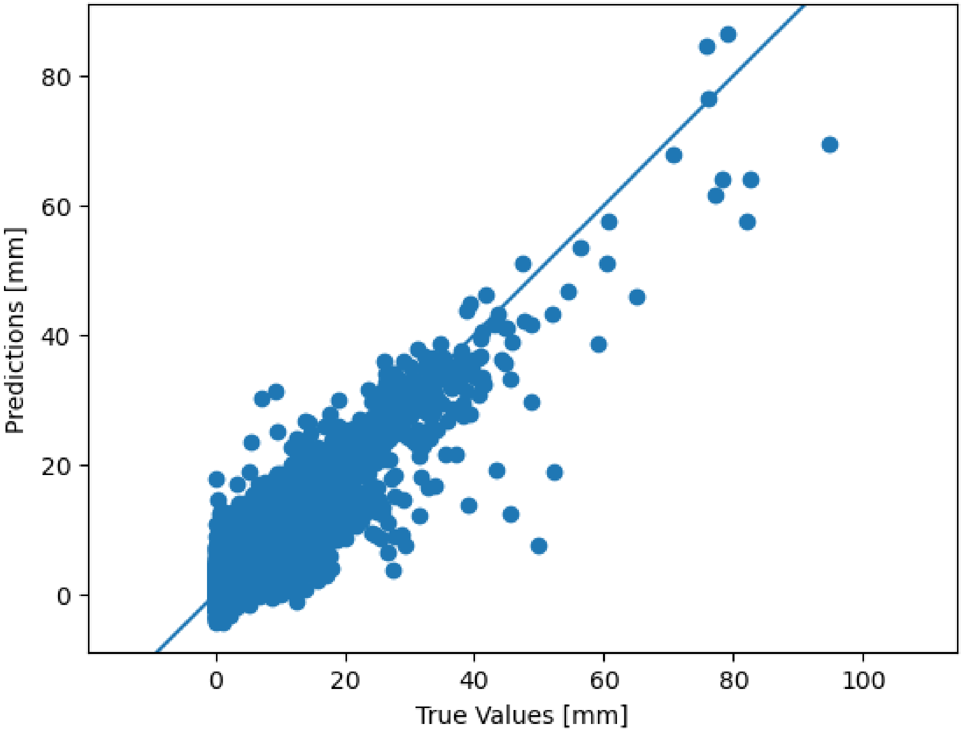
```
637/637 [==============================] - 0s 628us/step
Mean Absolute Error on Test Set: 1.1262069481053627 millimeters
```

In [24]:

```
test_predictions = model.predict(X_test).flatten()

plt.scatter(y_test, test_predictions)
plt.xlabel('True Values [mm]')
plt.ylabel('Predictions [mm]')
plt.axis('equal')
plt.xlim(plt.xlim())
plt.ylim(plt.ylim())
_ = plt.plot([-100, 100], [-100, 100])
```

```
637/637 [==============================] - 0s 581us/step
```



In [25]:

```
np.corrcoef(y_test,test_predictions)[0,1]
```

Out[25]:

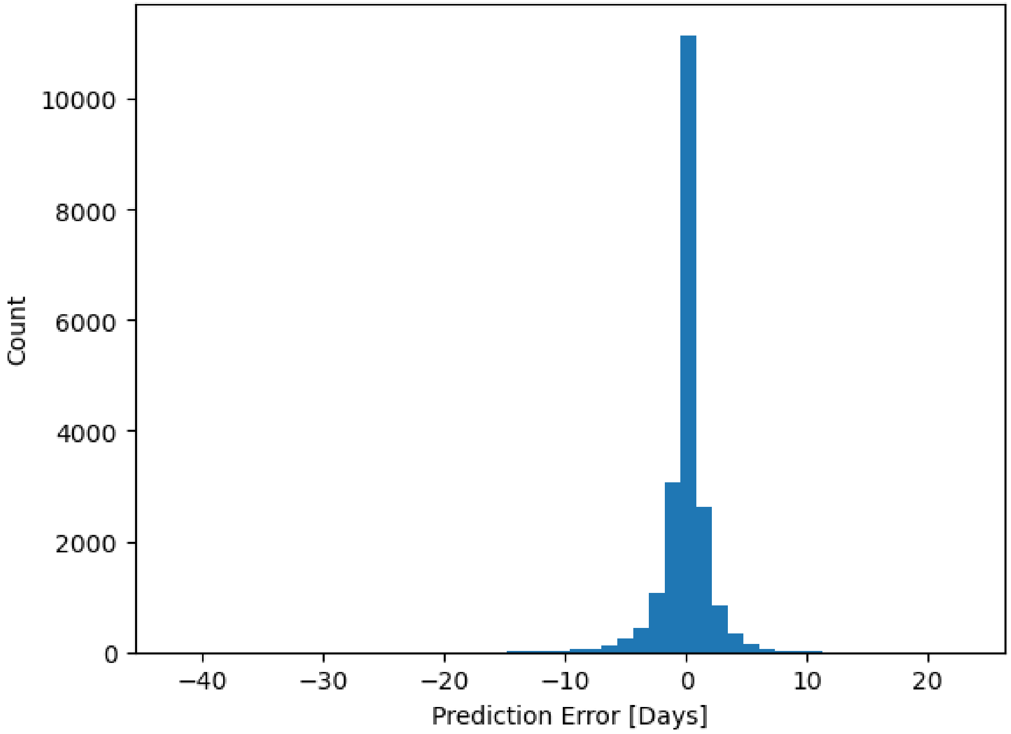
```
0.9211378742798222
```

In [26]:

```
from sklearn.metrics import r2_score
r2_score(y_test,test_predictions)
```

Out[26]:
```
0.8484118981514638
```

In [27]:
```
error = test_predictions - y_test
plt.hist(error, bins = 50)
plt.xlabel("Prediction Error [Days]")
_ = plt.ylabel("Count")
```



In [ ]:
```
#ATTEMPT for hyperparameter optimization and cross validation, require keras classifier bin
```

In [13]:
```
from sklearn.model_selection import GridSearchCV
```

In [14]:
```
# Step 3: Define the grid search parameters
param_grid_1 = dict(batch_size=[10, 40, 80], epochs=[10, 50])
print(param_grid_1)
```

```
{'batch_size': [10, 40, 80], 'epochs': [10, 50]}
```

In [15]:
```
# Step 4: Perform the grid search
grid_1 = GridSearchCV(estimator=model, param_grid=param_grid_1, n_jobs=1)
grid_result_1 = grid_1.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/jv/lz3tf5xn1vdbnrsqnz5bfz0r0000gn/T/ipykernel_1048/2948912329.py in <module>
      1 # Step 4: Perform the grid search
      2 grid_1 = GridSearchCV(estimator=model, param_grid=param_grid_1, n_jobs=1)
----> 3 grid_result_1 = grid_1.fit(train_data, train_labels)

~/opt/anaconda3/envs/keras/lib/python3.7/site-packages/sklearn/utils/validation.py in inner_f
(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
```