Deep Learning

Understanding Data

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
store_df = pd.read_csv('store.csv')
```

A	B	C	D	E	F	G	H	I
Store	DayOfWee	Date	Sales	Customers	Open	Promo	StateHolid	SchoolHoliday
1	5	7/31/2015	5263	555	1	1	0	1
2	5	7/31/2015	6064	625	1	1	0	1
3	5	7/31/2015	8314	821	1	1	0	1
4	5	7/31/2015	13995	1498	1	1	0	1

Store	StoreType	Assortmen	Competitic	Competitic	Competitic	Promo2	Promo2Sir	Promo2Sir	PromoInterval
1	c	a	1270	9	2008	0			
2	a	a	570	11	2007	1	13	2010	Jan,Apr,Jul,Oct
3	a	a	14130	12	2006	1	14	2011	Jan,Apr,Jul,Oct
4	c	c	620	9	2009	0			

Pre-Processing

Removing the column that are not in the test Data frame

```
#Dropping out Id coloumn to make it smoother
test_df.drop(['Id'],axis=1,inplace=True)
train_df.drop(["Customers"],axis=1,inplace=True)
```

Checking if there are any missing values in Train.CSV

```
# Checking for missing values in the training dataset
missing_values = train_df.isnull().sum()
missing_percentage = (missing_values / len(train_df)) * 100
missing_data_info = pd.DataFrame({'Missing_Values': missing_values, 'Missing_Percentage': missing_percentage})
missing_data_info = missing_data_info.sort_values(by='Missing_Percentage', ascending=False)
print("Columns with the highest percentage of missing values:")
print(missing_data_info.head())
```

```
Columns with the highest percentage of missing values:
           Missing_Values  Missing_Percentage
Store                   0                 0.0
DayOfWeek               0                 0.0
Date                    0                 0.0
Sales                   0                 0.0
Open                    0                 0.0
```

Pre-Processing

Checking if there are any missing values in test.CSV

```
Columns with the highest percentage of missing values:
           Missing_Values  Missing_Percentage
Open                   11            0.026772
Store                   0            0.000000
DayOfWeek               0            0.000000
Date                    0            0.000000
Promo                   0            0.000000
```

```python
test_df.fillna(0, inplace=True)
```

```
Columns with the highest percentage of missing values:
           Missing_Values  Missing_Percentage
Store                   0                 0.0
DayOfWeek               0                 0.0
Date                    0                 0.0
Open                    0                 0.0
Promo                   0                 0.0
```

Pre-Processing

Merging The data with the store data provided

```
train_store = pd.merge(train_df, store_df, on='Store')
test_store = pd.merge(test_df, store_df, on='Store')
train_store.head()
```

Checking if there are any missing values in Store.CSV

```
missing_values = train_store.isnull().sum()
```

```
Columns with the highest percentage of missing values:
                           Missing_Values  Missing_Percentage
PromoInterval                      508031           49.943620
Promo2SinceYear                    508031           49.943620
Promo2SinceWeek                    508031           49.943620
CompetitionOpenSinceYear           323348           31.787764
CompetitionOpenSinceMonth          323348           31.787764
```

Pre-Processing

Filling the Missing Values in the Store.CSV

After testing

- median
- mean
- zeros

```
train_store['CompetitionDistance'].fillna(train_store['CompetitionDistance'].median(), inplace=True)
test_store['CompetitionDistance'].fillna(test_store['CompetitionDistance'].median(), inplace=True)

train_store['PromoInterval'].fillna(0, inplace=True)
test_store['PromoInterval'].fillna(0, inplace=True)

train_store['CompetitionOpenSinceYear'].fillna(0, inplace=True)
train_store['CompetitionOpenSinceMonth'].fillna(0, inplace=True)
test_store['CompetitionOpenSinceYear'].fillna(0, inplace=True)
test_store['CompetitionOpenSinceMonth'].fillna(0, inplace=True)

train_store['Promo2SinceYear'].fillna(0, inplace=True)
train_store['Promo2SinceWeek'].fillna(0, inplace=True)
test_store['Promo2SinceYear'].fillna(0, inplace=True)
test_store['Promo2SinceWeek'].fillna(0, inplace=True)
```

Removing illogical Cases:

```
print("Closed stores with sales are ",((train_store["Open"]==0) & (train_store["Sales"]>0)).sum())
print("Open stores with zero sales are",((train_store["Open"]==1) & (train_store["Sales"]<=0)).sum())
print("negative sales are ",(train_store["Sales"]<0).sum())
```

```
Closed stores with sales are  0
Open stores with zero sales are 54
negative sales are  0
```

```
train_store = train_store[~((train_store["Open"] == 1) & (train_store["Sales"] <= 0))]
```

Pre-Processing

Removing Outliers

```
z_scores = np.abs(zscore(numeric))
outliers = (z_scores > 3)
print(f"Number of rows before removing outliers: {train_store.shape[0]}")
train_store = train_store[~np.any(outliers, axis=1)]
print(f"Number of rows after removing outliers: {train_store.shape[0]}")
```

```
Number of rows before removing outliers: 1017155
Number of rows after removing outliers: 986781
```

Encoding

Encoding Categorical Data

```
train_store['Date'] = pd.to_datetime(train_df['Date'])
test_store['Date'] = pd.to_datetime(test_df['Date'])

train_store['Week'] = pd.to_datetime(train_store['Date']).dt.isocalendar().week
train_store['DayOfYear'] = pd.to_datetime(train_store['Date']).dt.dayofyear
train_store['DayOfMonth'] = pd.to_datetime(train_store['Date']).dt.day
train_store['Year'] = pd.DatetimeIndex(train_store['Date']).year
train_store['Quarter'] = pd.DatetimeIndex(train_store['Date']).quarter
train_store['Month'] = pd.DatetimeIndex(train_store['Date']).month
```

```
train_store.drop(['Date'],axis=1,inplace=True)
```

Encoding and splitting Date data

Encoding

Encoding Categorical Data

```
#Manually encoding the stateholiday coloumn before using pre made encoders:
train_store['StateHoliday'].replace("a", 1, inplace=True)
train_store['StateHoliday'].replace("b", 2, inplace=True)
train_store['StateHoliday'].replace("c", 3, inplace=True)
train_store['StateHoliday'].replace('0', 0, inplace=True)
train_store['StateHoliday'].unique()
test_store['StateHoliday'].replace("a", 1, inplace=True)
test_store['StateHoliday'].replace("b", 2, inplace=True)
test_store['StateHoliday'].replace("c", 3, inplace=True)
test_store['StateHoliday'].replace('0', 0, inplace=True)
test_store['StateHoliday'].unique()
```

Manually Encoding

Encoding

Encoding Categorical Data

```
from sklearn.preprocessing import LabelEncoder
#try one hot encoder maybe better
category_train_store = train_store.select_dtypes(exclude=[np.number]).columns.tolist()
print("Categorical features in train dataset:", category_train_store)

Categorical_test_store = test_store.select_dtypes(exclude=[np.number]).columns.tolist()
print("Categorical features in test dataset:", Categorical_test_store)

le = LabelEncoder()
for column in category_train_store:
    train_store[column] = le.fit_transform(train_store[column].astype(str))

for column in Categorical_test_store:
    test_store[column] = le.fit_transform(test_store[column].astype(str))
```

```
Categorical features in train dataset: ['StoreType', 'Assortment', 'PromoInterval']
Categorical features in test dataset: ['StoreType', 'Assortment', 'PromoInterval']
```

Using Label Encoder

Tried Onehot encoder as well and get dummies

Scaling

Normalizing Data Using MinMax

```
scaler = MinMaxScaler()
train_scaling=train_store.drop(["Sales"],axis=1,inplace=False)
scaled_x_train_store = pd.DataFrame(scaler.fit_transform(train_scaling),columns=train_scaling.columns)
scaled_test_store = scaler.fit_transform(test_store)
scaled_y_train_store = pd.DataFrame(scaler.fit_transform(pd.DataFrame(train_store["Sales"])),columns=["Sales"])
scaled_train_store=pd.concat([scaled_x_train_store, scaled_y_train_store], axis=1)
scaled_test_store = pd.DataFrame(scaled_test_store, columns=test_store.columns)
```

```
print(scaled_train_store.head())
```

```
      Store  DayOfWeek  Open  Promo  StateHoliday  SchoolHoliday  StoreType  \
0  0.000000   0.666667   1.0    1.0           0.0            1.0   0.666667
1  0.000898   0.666667   1.0    1.0           0.0            1.0   0.000000
2  0.001795   0.666667   1.0    1.0           0.0            1.0   0.000000
3  0.002693   0.666667   1.0    1.0           0.0            1.0   0.666667
4  0.004488   0.666667   1.0    1.0           0.0            1.0   0.000000

   Assortment  CompetitionDistance  CompetitionOpenSinceMonth  ...  \
0         0.0             0.045241                   0.750000  ...
1         0.0             0.019906                   0.916667  ...
2         0.0             0.510677                   1.000000  ...
3         1.0             0.021716                   0.750000  ...
4         0.0             0.010496                   1.000000  ...
```

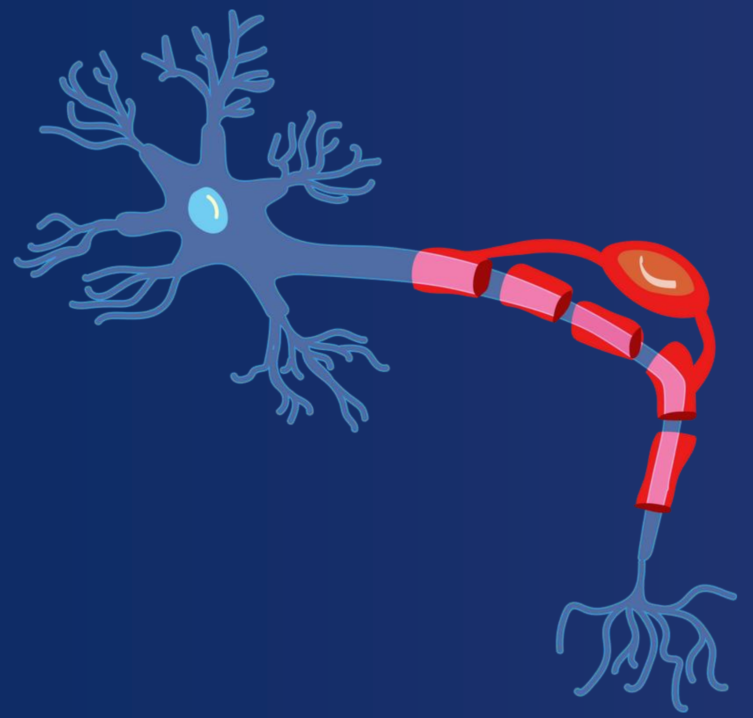
Splitting

Normalizing Data Using MinMax

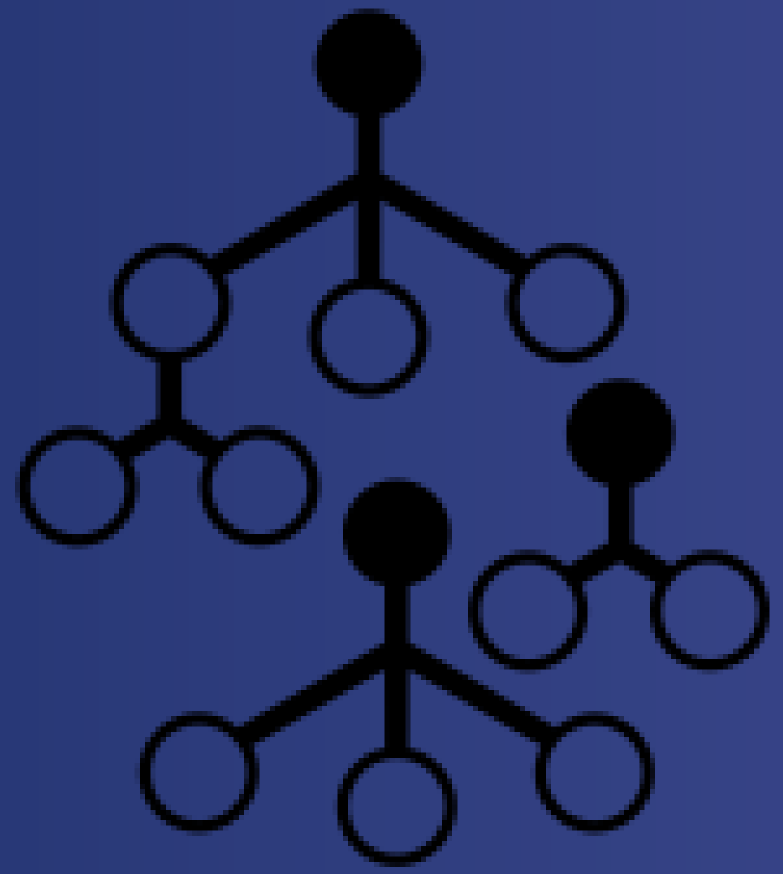
```
x=scaled_train_store.drop(['Sales'],axis=1)
y=scaled_train_store['Sales']
print(y)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=7)
```

Using 3 Different models

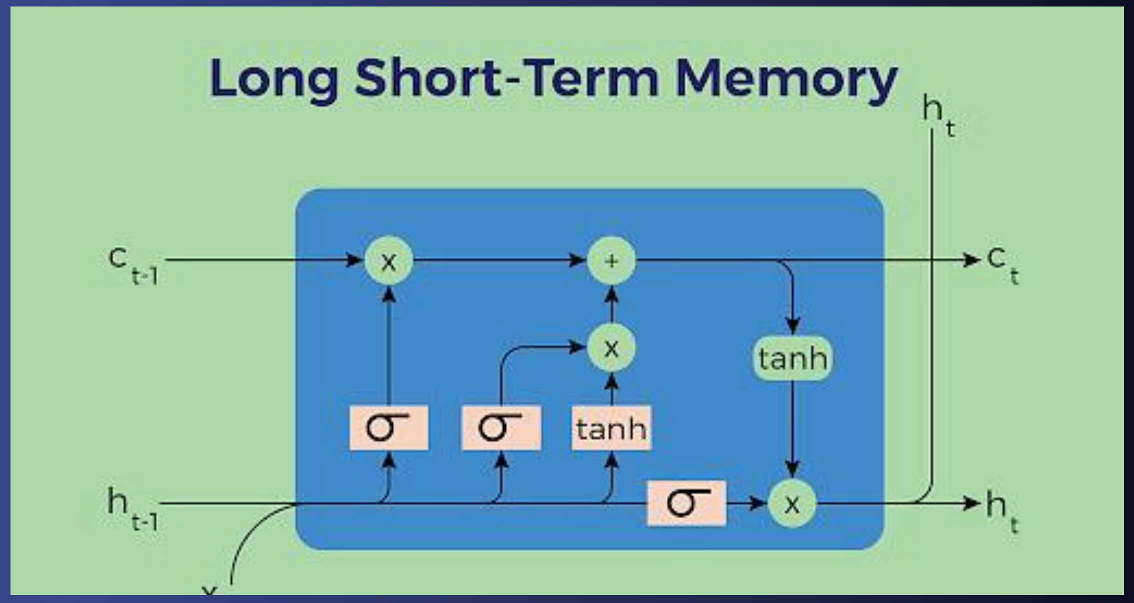
FCNN



ML algorithms



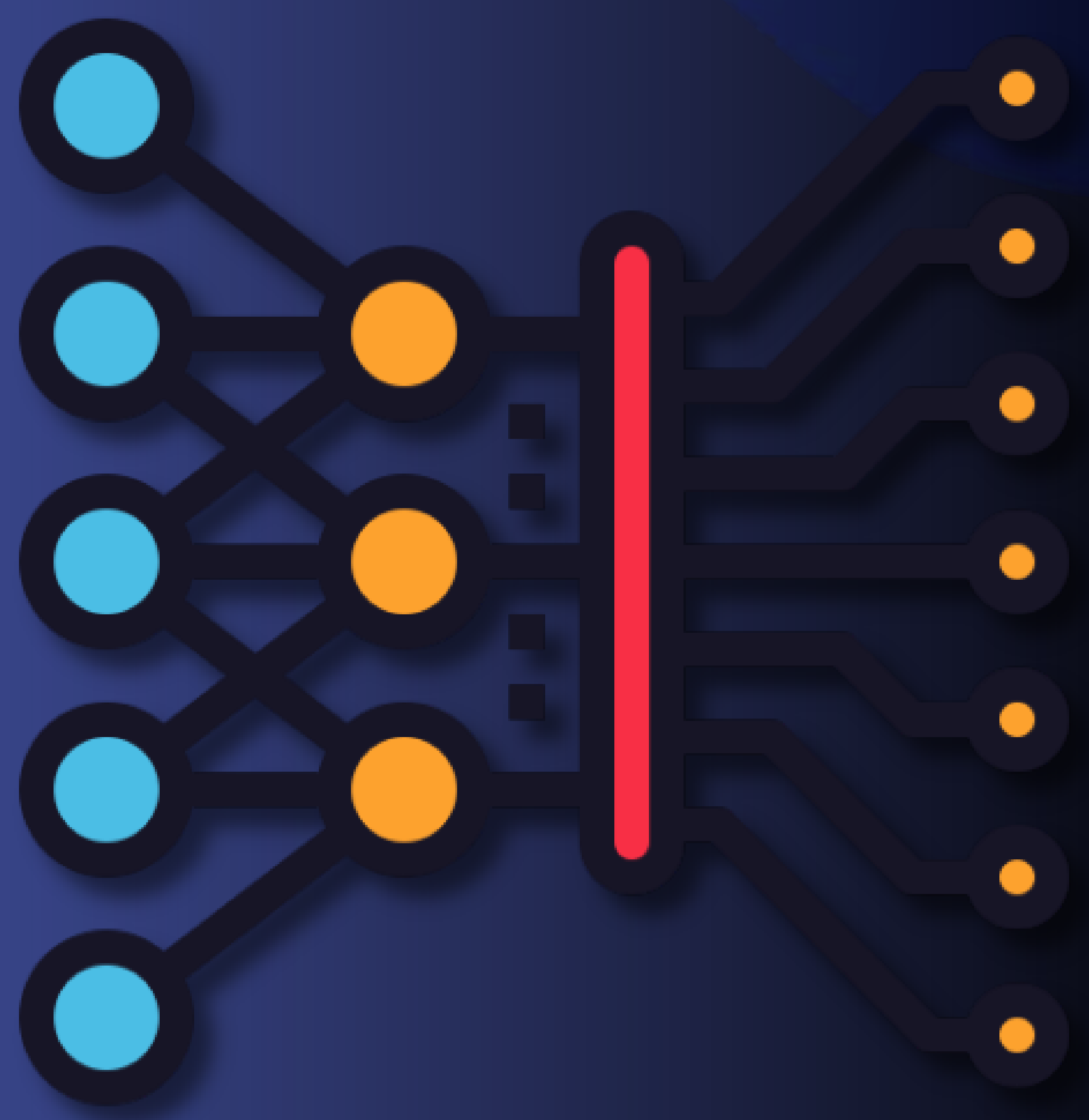
LSTM



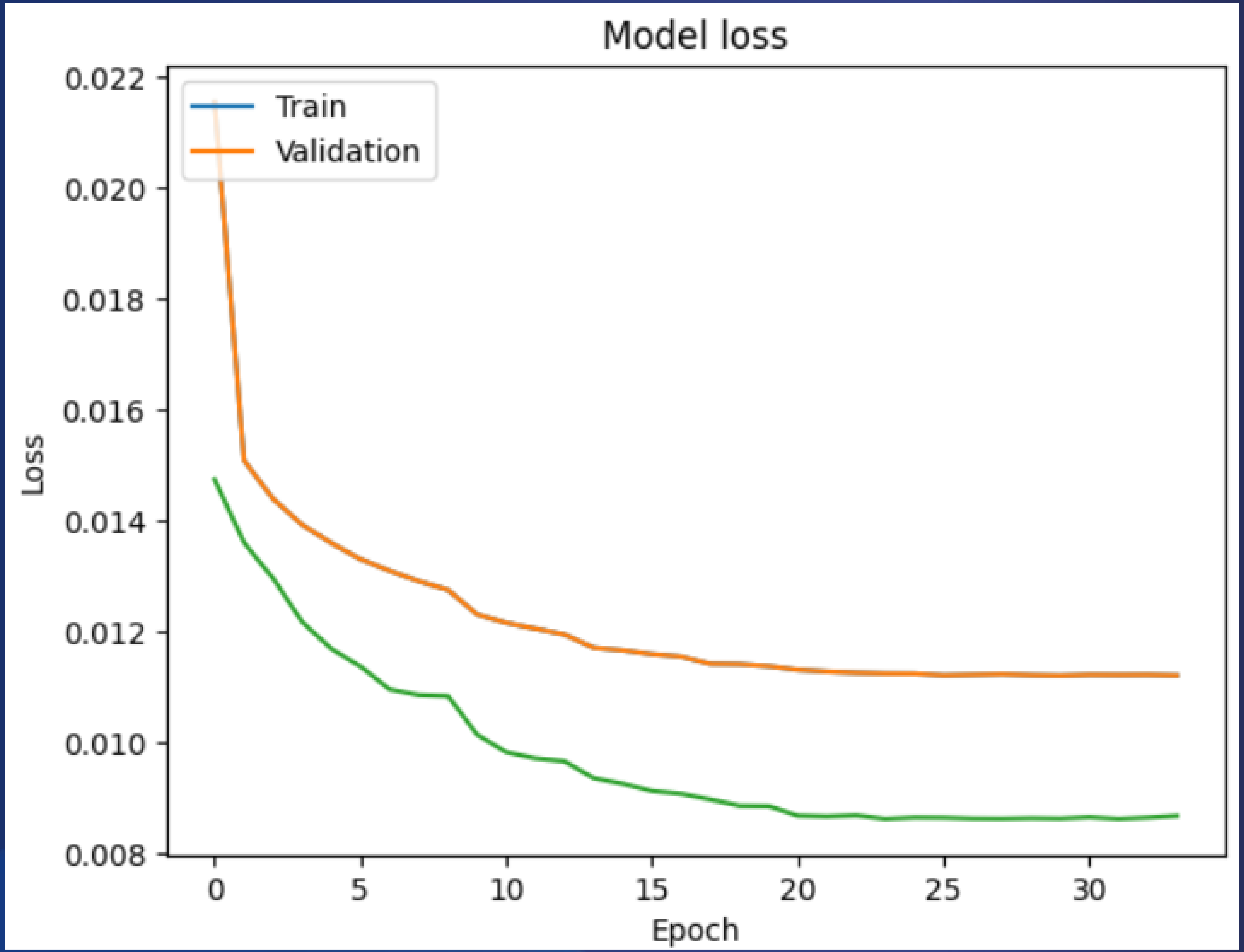
FCNN

```
model = models.Sequential([
    layers.Input(shape=(x_train.shape[1],)),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),  #s
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)
])
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1)
]
# Compile model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.summary()

# Train the model
batch_size = 32
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=70, callbacks=callbacks, batch_size=batch_size)
```



FCNN



Model loss
0.022
0.020
0.018
0.016
0.014
0.012
0.010
0.008
Loss
Train
Validation
0
5
10
15
20
25
30
Epoch

ML Algorithms

```
models = [
    ('Random Forest', RandomForestRegressor()),
    ('Ridge Regression', Ridge()),
    ('Support Vector Machine', SVR()),
    ('Neural Network', MLPRegressor())
]
# Hyperparameters for grid search (you can modify these for each model)
param_grid = {
    'Random Forest': {'n_estimators': [100, 300, 500]},
    'Ridge Regression': {'alpha': [0.1, 1.0, 10.0]},
    'Support Vector Machine': {'kernel': ['linear', 'rbf'], 'C': [1, 10]},
    'Neural Network': {'hidden_layer_sizes': [(50,), (100,)], 'alpha': [0.0001, 0.001]}
}
```

```
for model_name, model in models:
    grid_search = GridSearchCV(model, param_grid[model_name], scoring='neg_mean_squared_error', cv=5)
    grid_search.fit(X_train, y_train)
    y_pred = grid_search.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    if mse < best_score:
        best_score = mse
        best_model = grid_search.best_estimator_
        best_model_name = model_name

    print(f"{model_name} - Mean Squared Error: {mse}")
print(f"Best Model: {best_model_name} - Best Mean Squared Error: {best_score}")
```

ML Algorithms

```
Random Forest - Mean Squared Error: 0.0005290457524361835
Ridge Regression - Mean Squared Error: 0.004696394952530777
```

LSTM

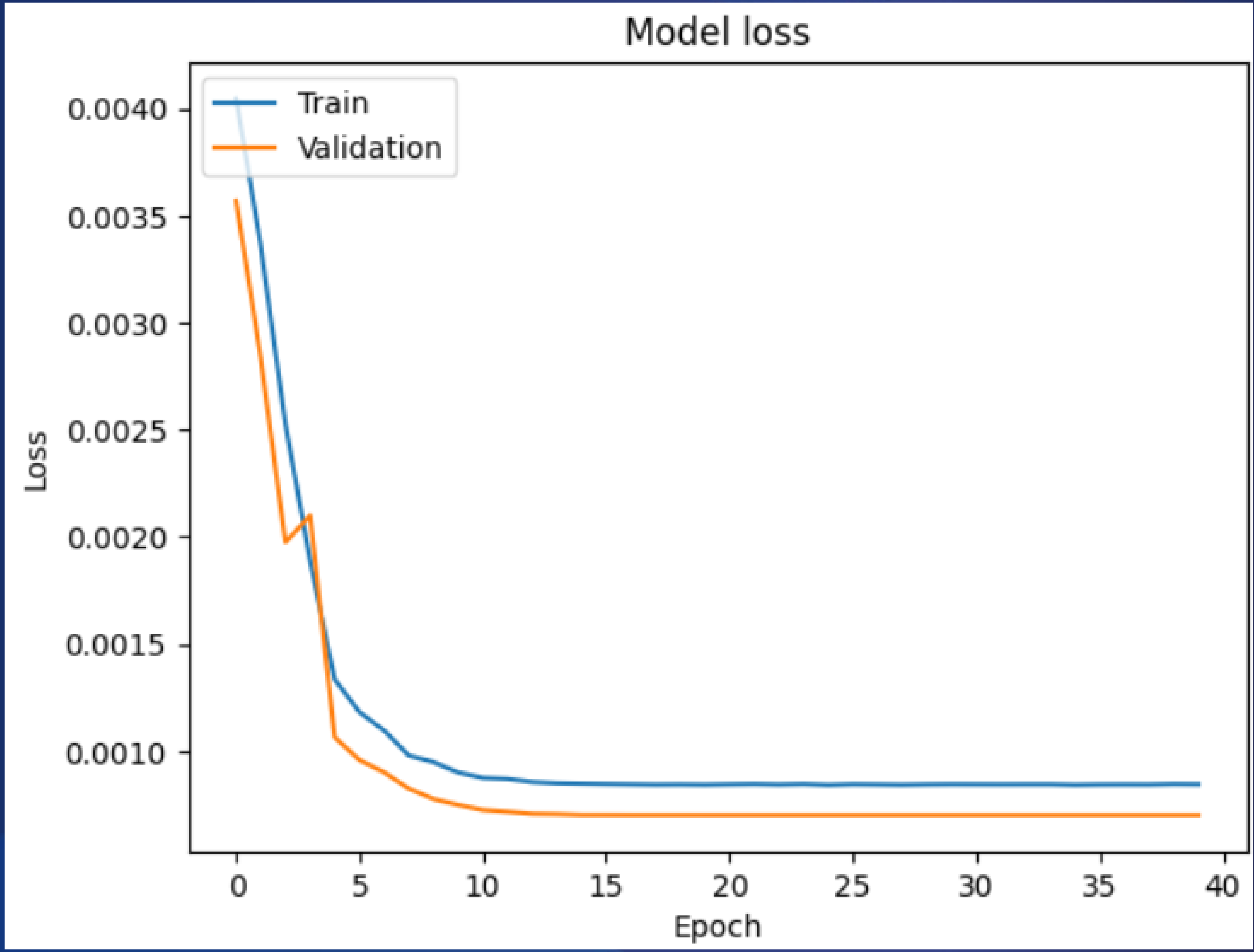
```
model = models.Sequential([
    layers.LSTM(64, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])),
    layers.Dropout(0.2),
    layers.LSTM(32),
    layers.Dropout(0.2),
    layers.Dense(1)
])
```

```
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1)
]

# Compile model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.summary()

# Train the model
batch_size = 32
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=20, callbacks=callbacks, batch_size=batch_size)
```

LSTM


Model loss
Train
Validation
Loss
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0
5
10
15
20
25
30
35
40
Epoch

Testing the Model

```
6358/6358 ━━━━━━━━━━━━━━━━━━━━ 19s 3ms/step - loss: 6.9387e-04 - mae: 0.0168
Test Mean Absolute Error: 0.01686321571469307
1284/1284 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
Predicted Sales: [[ 6260.2036]
 [ 8129.6177]
 [10756.432 ]
 ...
 [ 5460.703 ]
 [21708.084 ]
 [ 5102.3276]]
          ID         Sales
0          1   6260.203613
1          2   8129.617676
2          3  10756.431641
3          4   7594.935547
4          5   7585.748535
...      ...           ...
41083  41084   3141.491455
41084  41085   8225.785156
41085  41086   5460.703125
41086  41087  21708.083984
41087  41088   5102.327637

[41088 rows x 2 columns]
```

Thank You!