```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from tensorflow import keras
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from tensorflow.keras import callbacks
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Conv1D, Conv2D, MaxPooling1D, MaxPooling2D, Flatten, LSTM, Dense, TimeDistributed, Dropout, BatchN
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import backend as K
from scikeras.wrappers import KerasRegressor
import torch
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
import scipy
import os
import random
import math

seed = 42
os.environ['PYTHONHASHSEED']=str(seed)
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

Import needed files for spatial analysis:

- House Listings scraped from Trulia
- Location data for all Arizona supermarkets
- Location data for all Arizona stores
- Arizona population data by city

```
trulia_df = pd.read_csv('/content/trulia_data.csv')
```

```
stores_df_raw = pd.read_csv('/content/Arizona_stores_latlong.csv')
stores_df = stores_df_raw.reset_index()
```

```
schools_df = pd.read_excel('/content/Schools.xlsx')
```

```
pop_df = pd.read_csv('/content/us-cities-table-for-arizona.csv')
```

Clean data: remove missing values, create numeric versions of variables, etc

```
trulia_df.isna().sum()
```

```
address              0
city                 0
state                0
zipcode              0
latitude             0
longitude            0
price                2
link                 0
floor_space         45
beds                37
baths              109
year_built          35
parking            485
price_per_sqft      46
HOA                936
heating           1079
cooling           1079
outdoor           1065
```

```
pool               266
laundry           1079
listing_agent      845
broker             290
date                 6
date_type            6
sold_indicator       0
dtype: int64
```

```
trulia_df_notnull = trulia_df[trulia_df['price'].notna() & trulia_df['date'].notna() & trulia_df['year_built'].notna() & trulia_df['floor_sp
```

```
price_num = []
for h in trulia_df_notnull['price']:
    price_num += [float(h.replace('$','').replace(',',''))]
trulia_df_notnull['price_num'] = price_num
date_str = []
year = []
for i in trulia_df_notnull['date']:
    year += [int(i.split('-')[0])]
    if str(i).find('T') == -1:
        date_str += [str(i) + 'T00:00:00+00:00']
    else:
        date_str += [str(i)]
trulia_df_notnull['date_str'] = date_str
trulia_df_notnull['year'] = year
trulia_df_notnull['date_format'] = pd.to_datetime(trulia_df_notnull['date_str'])
trulia_df_notnull['house_age'] = np.where(trulia_df_notnull['year_built'].notna(),trulia_df_notnull['year']-trulia_df_notnull['year_built'],
floor_space_str = []
for j in trulia_df_notnull['floor_space']:
    floor_space_str += [str(j)]
trulia_df_notnull['floor_space_str'] = floor_space_str
floor_space_num = []
for k in trulia_df_notnull['floor_space_str']:
    if k == 'nan':
        floor_space_num += [np.nan]
    else:
        floor_space_num += [float(k.replace(',','').replace('sqft','').strip())]
trulia_df_notnull['floor_space_num'] = floor_space_num
beds_num = []
for l in trulia_df_notnull['beds']:
    if str(l) == 'nan':
        beds_num += [np.nan]
    elif str(l) == 'Studio':
        beds_num += [0]
    else:
        beds_num += [int(l[:1])]
trulia_df_notnull['beds_num'] = beds_num
baths_num = []
for m in trulia_df_notnull['baths']:
    if str(m) == 'nan':
        baths_num += [np.nan]
    elif str(m) == 'Studio':
        baths_num += [0]
    else:
        baths_num += [int(m[:1])]
trulia_df_notnull['baths_num'] = baths_num
trulia_df_notnull['garage_ind'] = np.where(trulia_df_notnull['parking']=='Garage',1,0)
price_per_sqft_num = []
for n in trulia_df_notnull['price_per_sqft']:
    if str(n) == 'nan':
        price_per_sqft_num += [np.nan]
    else:
        price_per_sqft_num += [float(str(n).replace(',','').replace('$','').replace('/sqft','').strip())]
trulia_df_notnull['price_per_sqft_num'] = price_per_sqft_num
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['date_str'] = date_str
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['date_format'] = pd.to_datetime(trulia_df_notnull['date_str'])
<ipython-input-9-825ed74b0857>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['house_age'] = np.where(trulia_df_notnull['year_built'].notna(),trulia_df_notnull['year']-trulia_df_notnull['year
<ipython-input-9-825ed74b0857>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['floor_space_str'] = floor_space_str
<ipython-input-9-825ed74b0857>:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['floor_space_num'] = floor_space_num
<ipython-input-9-825ed74b0857>:36: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['beds_num'] = beds_num
<ipython-input-9-825ed74b0857>:45: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['baths_num'] = baths_num
<ipython-input-9-825ed74b0857>:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  trulia_df_notnull['garage_ind'] = np.where(trulia_df_notnull['parking']=='Garage',1,0)
<ipython-input-9-825ed74b0857>:53: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
```

Merge all needed datasets together into one spatial dataset and compute distances from houses to a store/school

```
trulia_merge_pop = trulia_df_notnull.merge(pop_df, how='outer', left_on=['city'], right_on=['name'])
trulia_merge_pop = trulia_merge_pop[trulia_merge_pop['price'].notna() & trulia_merge_pop['pop2024'].notna()].drop(columns=['pop2020','growth
trulia_merge_pop = trulia_merge_pop.reset_index()
```

```
#Haversine function to compute distance
def haversine(latA,longA,latB,longB):
    latA_rad = math.radians(latA)
    longA_rad = math.radians(longB)
    latB_rad = math.radians(latB)
    longB_rad = math.radians(longB)
    step1 = (math.sin((latA_rad - latB_rad)/2)**2) + math.cos(latA_rad)*math.cos(latB_rad)*(math.sin((longA_rad - longB_rad)/2)**2)
    step2 = 2*math.atan2(math.sqrt(step1), math.sqrt(1 - step1))
    step3 = 6371*step2 #multiply by Earth's circumference in km
    return step3
```

```
minSchoolDistFinal = pd.DataFrame(columns=['homeAddress','min_school_dist','DistrictEntityID','SchoolEntityID','DistrictName','SchoolName','
for row in range(len(trulia_merge_pop)):
    #print(trulia_merge_pop.loc[row]['latitude'])
    #print(trulia_merge_pop.loc[row]['longitude'])
    school_dist = {}
    for school in range(len(schools_df)):
        #print(schools_df.loc[school]['lon'])
        dist = haversine(trulia_merge_pop.loc[row]['latitude'],trulia_merge_pop.loc[row]['longitude'],schools_df.loc[school]['lat'],schools_
        school_dist[dist] = {'DistrictEntityID':schools_df.loc[school]['DistrictEntityID'],'SchoolEntityID':schools_df.loc[school]['SchoolEn
    minSchoolDist = min(school_dist.keys())
    #print(type(school_dist[minSchoolDist]))
    minSchoolDist_df = pd.DataFrame(school_dist[minSchoolDist],index=[row])
    minSchoolDist_df['homeAddress'] = trulia_merge_pop.loc[row]['address']
    minSchoolDist_df['min_school_dist'] = minSchoolDist
    minSchoolDistFinal = pd.concat([minSchoolDistFinal,minSchoolDist_df])
```

```
minStoreDistFinal = pd.DataFrame(columns=['homeAddress','min_store_dist','store_name','store_address','store_city','store_state','store_zipc
for row in range(len(trulia_merge_pop)):
    store_dist = {}
    for store in range(len(stores_df)):
        dist = haversine(trulia_merge_pop.loc[row]['latitude'],trulia_merge_pop.loc[row]['longitude'],stores_df.loc[store]['lat'],stores_df.
        store_dist[dist] = {'store_name':stores_df.loc[store]['name'],'store_address':stores_df.loc[store]['address'],'store_city':stores_df
    minStoreDist = min(store_dist.keys())
    minStoreDist_df = pd.DataFrame(store_dist[minStoreDist],index=[0])
    minStoreDist_df['homeAddress'] = trulia_merge_pop.loc[row]['address']
    minStoreDist_df['min_store_dist'] = minStoreDist
    minStoreDistFinal = pd.concat([minStoreDistFinal,minStoreDist_df])


data_merge_school_raw = trulia_merge_pop.merge(minSchoolDistFinal, how='outer', left_on=['address'], right_on=['homeAddress'])
data_merge_school = data_merge_school_raw[data_merge_school_raw['price'].notna()].drop_duplicates()
data_merge_store_raw = data_merge_school.merge(minStoreDistFinal, how='outer', left_on=['address'], right_on=['homeAddress'])
data_merge_store = data_merge_store_raw[data_merge_store_raw['price'].notna()].drop_duplicates()
```

Save data up to this point to csv file

```
data_merge_store.to_csv('data_merge_store.csv')
```

```
data_merge_store = pd.read_csv('/content/data_merge_store.csv')
```

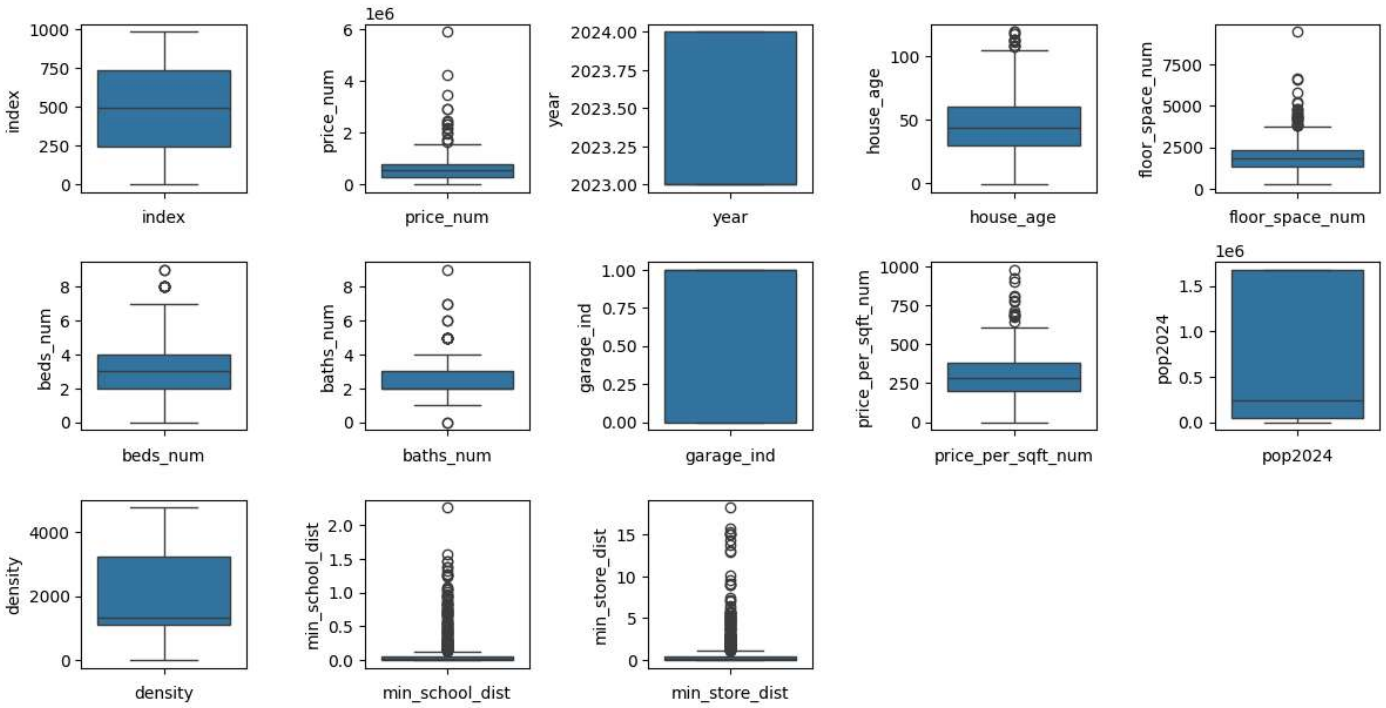
Keep only needed columns, filter out outliers, and examine variable distributions with boxplots.

```
final_df = data_merge_store[['price_num','city','year','house_age','floor_space_num','beds_num','baths_num','garage_ind','price_per_sqft_num
```

```
final_df_rem_out = final_df[(final_df['price_num'] < 10000000) & (final_df['price_per_sqft_num'] < 1000) & (final_df['floor_space_num'] < 10
final_df_rem_out = final_df_rem_out.reset_index()
```

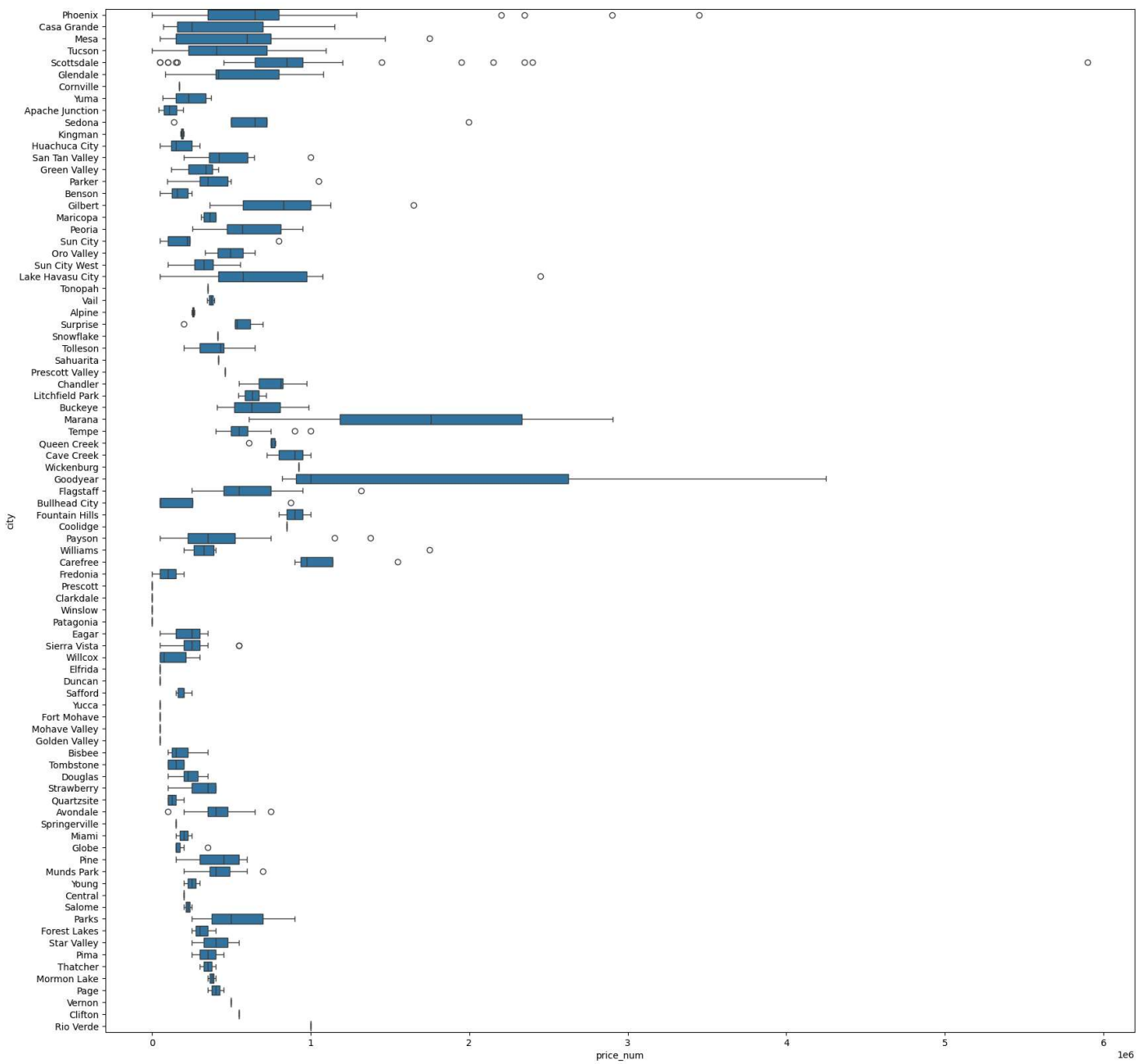
```
plt.figure(figsize = (12,12))

for i,j in enumerate(final_df_rem_out.select_dtypes(include = "number").columns):

    plt.subplot(6,5,i+1)

    sns.boxplot(final_df_rem_out[j])
    plt.xlabel("{}".format(j))

    plt.tight_layout()

plt.subplots_adjust()
plt.show()
```

[1]




```
final_df_small = final_df_rem_out[['price_num','city','year','house_age','garage_ind','price_per_sqft_num','pop2024','density','floor_space_
```

```
plt.figure(figsize = (20,20))
sns.boxplot(data=final_df_small,x='price_num',y='city')
plt.show()
```

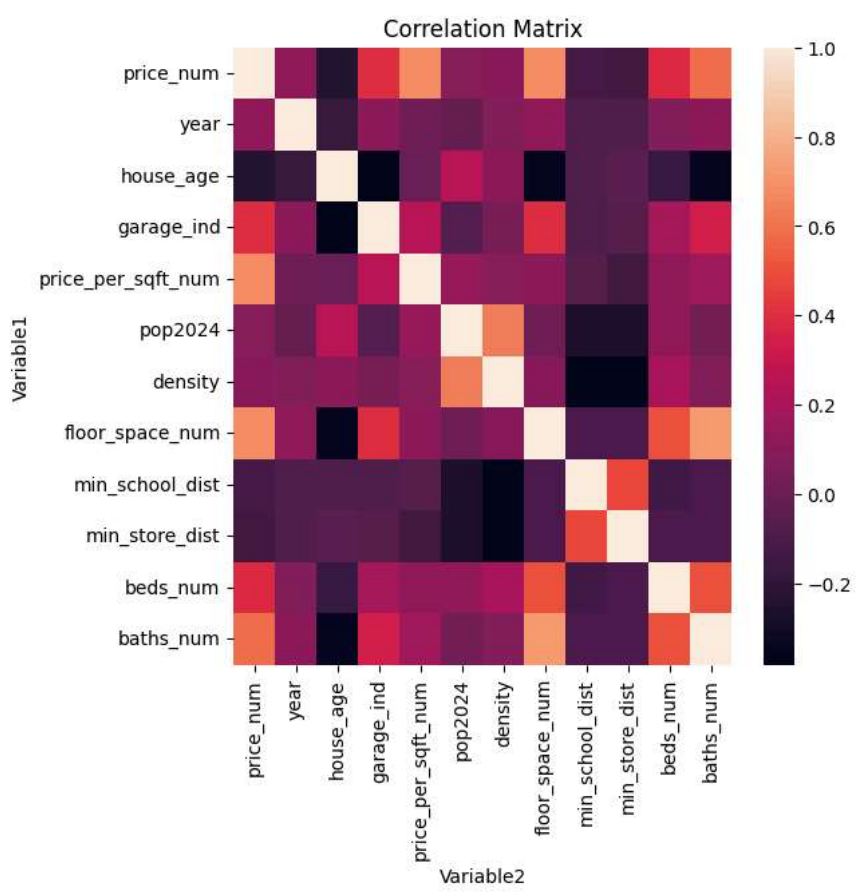


Correlation Analysis

```
corr_all_matrix = final_df_small.select_dtypes(include = "number").corr()

corr_all_df = corr_all_matrix.stack().reset_index()
corr_all_df.columns = ['v1', 'v2', 'correlation']

corr_all_df_sort = corr_all_df.loc[corr_all_df.correlation.abs().sort_values(ascending= False).index]
corr_all_df_sort.loc[(corr_all_df_sort.v1 == 'price_num')]
```

	v1	v2	correlation
0	price_num	price_num	1.000000
4	price_num	price_per_sqft_num	0.685061
7	price_num	floor_space_num	0.672881
11	price_num	baths_num	0.579883
3	price_num	garage_ind	0.397193
10	price_num	beds_num	0.384743
2	price_num	house_age	-0.241299
9	price_num	min_store_dist	-0.138965
1	price_num	year	0.125751
8	price_num	min_school_dist	-0.112845
6	price_num	density	0.101681
5	price_num	pop2024	0.091256

```
plt.figure(figsize = (6,6))
sns.heatmap(corr_all_matrix)
plt.title('Correlation Matrix')
plt.ylabel('Variable1')
plt.xlabel('Variable2')
plt.show()
```



Data Imputation

```
final_df_small.isna().sum()
```

```
price_num              0
city                   0
year                   0
house_age              0
garage_ind             0
price_per_sqft_num     0
pop2024                0
density                0
floor_space_num        0
min_school_dist        0
min_store_dist         0
beds_num               6
baths_num             57
dtype: int64
```

```
minimum = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['beds_num']).min()
median = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['beds_num']).median()
maximum = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['beds_num']).max()
mean = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['beds_num']).mean()
pd.concat([minimum,median,maximum,mean],axis=1)
```

beds_num	floor_space_num	floor_space_num	floor_space_num	floor_space_num
0.0	324	1480.0	9461	1576.254098
1.0	400	676.0	1232	716.074074
2.0	264	1192.0	3598	1299.000000
3.0	672	1739.0	4184	1838.787466
4.0	1344	2266.0	6648	2420.555085
5.0	1491	2802.0	5810	3000.489796
6.0	1938	2787.0	6566	3264.937500
7.0	2450	2450.0	2450	2450.000000
8.0	1000	3200.0	3696	2952.400000
9.0	3468	4159.0	4850	4159.000000

```
minimum = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['baths_num']).min()
median = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['baths_num']).median()
maximum = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['baths_num']).max()
mean = final_df_rem_out['floor_space_num'].groupby(final_df_rem_out['baths_num']).mean()
pd.concat([minimum,median,maximum,mean],axis=1)
```

baths_num	floor_space_num	floor_space_num	floor_space_num	floor_space_num
0.0	480	502.0	524	502.000000
1.0	264	897.0	3696	1078.301587
2.0	530	1690.5	3134	1718.478903
3.0	1278	2376.5	6648	2438.396552
4.0	1650	3199.5	5146	3125.828571
5.0	2100	3966.0	4804	3856.100000
6.0	5208	5509.0	5810	5509.000000
7.0	1000	3783.0	6566	3783.000000
9.0	3468	3468.0	3468	3468.000000

```
bed_update = []
for rec in range(len(final_df_rem_out)):
    if np.isnan(final_df_rem_out.loc[rec]['beds_num']) == False:
        bed_update += [final_df_rem_out.loc[rec]['beds_num']]
    elif final_df_rem_out.loc[rec]['floor_space_num'] < 900:
        bed_update += [1]
    elif 900 <= final_df_rem_out.loc[rec]['floor_space_num'] < 1500:
        bed_update += [2]
    elif 1500 <= final_df_rem_out.loc[rec]['floor_space_num'] < 2000:
        bed_update += [3]
    elif 2000 <= final_df_rem_out.loc[rec]['floor_space_num'] < 2550:
        bed_update += [4]
    else:
        bed_update += [5]
bath_update = []
for rec in range(len(final_df_rem_out)):
    if np.isnan(final_df_rem_out.loc[rec]['baths_num']) == False:
        bath_update += [final_df_rem_out.loc[rec]['baths_num']]
    elif final_df_rem_out.loc[rec]['floor_space_num'] < 1300:
        bath_update += [1]
    elif 1300 <= final_df_rem_out.loc[rec]['floor_space_num'] < 2050:
        bath_update += [2]
    elif 2050 <= final_df_rem_out.loc[rec]['floor_space_num'] < 2800:
        bath_update += [3]
    elif 2800 <= final_df_rem_out.loc[rec]['floor_space_num'] < 3550:
        bath_update += [4]
    elif 3550 <= final_df_rem_out.loc[rec]['floor_space_num'] < 5000:
        bath_update += [5]
    else:
        bath_update += [6]

final_df_impute = final_df_rem_out.drop(columns=['beds_num','baths_num'])
final_df_impute['beds_num'] = bed_update
final_df_impute['baths_num'] = bath_update
```

Save final cleaned spatial data file

```
final_df_impute[['price_num','city','year','house_age','garage_ind','price_per_sqft_num','pop2024','density','floor_space_num','min_school_d


spatial_data_raw = pd.read_csv('/content/final_housing_data.csv')
spatial_data = spatial_data_raw[['price_num','city','year','house_age','garage_ind','price_per_sqft_num','pop2024','density','floor_space_nu
display(spatial_data)
```

	price_num	city	year	house_age	garage_ind	price_per_sqft_num	pop2024	density	floor_space_num	min_school_dist	min_store_d
0	185000	Phoenix	2024	51	1	168	1676481	3234	1100	0.012764	0.002
1	169900	Phoenix	2024	61	1	242	1676481	3234	702	0.018489	0.021
2	99500	Phoenix	2024	53	0	66	1676481	3234	1512	0.027269	0.164
3	175000	Phoenix	2024	17	0	311	1676481	3234	563	0.023884	0.107
4	165000	Phoenix	2024	54	0	197	1676481	3234	836	0.005792	0.331
...	...	...	...	...	...	...	...	...	...	...	
971	350000	Page	2023	46	1	247	7297	191	1419	0.044256	0.105
972	450000	Page	2024	63	0	209	7297	191	2158	0.042699	0.020
973	500000	Vernon	2023	31	1	175	317	560	2850	0.114642	2.684
974	550000	Clifton	2023	33	0	277	3627	247	1987	0.005508	0.820
975	1000000	Rio Verde	2024	25	1	349	2458	568	2863	0.086644	0.782

976 rows × 13 columns

	price_num	city	year	house_age	garage_ind	price_per_sqft_num	pop2024	density	floor_space_num	min_school_dist	min_store_d
0	185000	Phoenix	2024	51	1	168	1676481	3234	1100	0.012764	0.002
1	169900	Phoenix	2024	61	1	242	1676481	3234	702	0.018489	0.021
2	99500	Phoenix	2024	53	0	66	1676481	3234	1512	0.027269	0.164
3	175000	Phoenix	2024	17	0	311	1676481	3234	563	0.023884	0.107
4	165000	Phoenix	2024	54	0	197	1676481	3234	836	0.005792	0.331
...	...	...	...	...	...	...	...	...	...	...	
971	350000	Page	2023	46	1	247	7297	191	1419	0.044256	0.105
972	450000	Page	2024	63	0	209	7297	191	2158	0.042699	0.020
973	500000	Vernon	2023	31	1	175	317	560	2850	0.114642	2.684
974	550000	Clifton	2023	33	0	277	3627	247	1987	0.005508	0.820
975	1000000	Rio Verde	2024	25	1	349	2458	568	2863	0.086644	0.782

Create label encoded city variable

```
allCities = spatial_data['city'].unique()
label_encoder = LabelEncoder()
label_encoder.fit(allCities)
spatial_data['cityEncoded'] = label_encoder.transform(spatial_data['city'])
spatial_data_float = spatial_data.drop(['city'], axis=1)
spatial_data_float = spatial_data_float.astype('float32')
```

Split data into training and test sets

```
targetCNN = spatial_data_float['price_num']
featuresCNN = spatial_data_float.drop(['price_num'], axis=1)

X_train, X_other, y_train, y_other = train_test_split(featuresCNN, targetCNN, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_other, y_other, test_size=0.5, random_state=42)
```

Employ baseline models

```
#Linear Model
lr = LinearRegression()
lr.fit(X_train,y_train)
coefficients = pd.concat([pd.DataFrame(X_test.columns),pd.DataFrame(np.transpose(lr.coef_))],axis=1)
coefficients
```

	0	0
0	year	38081.121094
1	house_age	-188.823029
2	garage_ind	31569.755859
3	price_per_sqft_num	1729.743896
4	pop2024	0.014343
5	density	-10.926692
6	floor_space_num	245.897552
7	min_school_dist	-21170.144531
8	min_store_dist	1558.612549
9	beds_num	11773.459961
10	baths_num	-1975.088135
11	cityEncoded	-399.205048

```
lr_pred = lr.predict(X_test)
lr_mse = mean_squared_error(y_test,lr_pred)
lr_mse
```

```
16279484000.0
```

```
#Drop some fields and create new linear model
X_train_drop = X_train.drop(columns=['pop2024','density'])
X_test_drop = X_test.drop(columns=['pop2024','density'])
lr_drop = LinearRegression()
lr_drop.fit(X_train_drop,y_train)
coefficients_lrdrop = pd.concat([pd.DataFrame(X_test_drop.columns),pd.DataFrame(np.transpose(lr_drop.coef_))],axis=1)
coefficients_lrdrop
```

	0	0
0	year	34588.906250
1	house_age	-168.333008
2	garage_ind	30593.935547
3	price_per_sqft_num	1734.169800
4	floor_space_num	245.475159
5	min_school_dist	-14427.643555
6	min_store_dist	2672.183838
7	beds_num	10936.528320
8	baths_num	-1183.266235
9	cityEncoded	-392.129395

```
lr_drop_pred = lr_drop.predict(X_test_drop)
lr_drop_mse = mean_squared_error(y_test,lr_drop_pred)
lr_drop_mse
```

```
16389855000.0
```

```
#Random Forest Regressor
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train,y_train)
rfr_cv_scores = cross_val_score(rfr, X_train, y_train, scoring='neg_mean_squared_error', cv=10)
rfr_mse_scores = -rfr_cv_scores
rfr_mse_scores
```

```
array([1.08120592e+09, 8.87954741e+08, 6.50926379e+09, 6.40951556e+10,
       1.52885854e+09, 1.03660222e+09, 1.14952625e+10, 3.18019507e+09,
       2.96744727e+10, 1.40022767e+09])
```

```
#Second random forest regressor
rfr_drop = RandomForestRegressor(random_state=42)
rfr_drop.fit(X_train_drop,y_train)
rfr_drop_cv_scores = cross_val_score(rfr_drop, X_train_drop, y_train, scoring='neg_mean_squared_error', cv=10)
rfr_drop_mse_scores = -rfr_drop_cv_scores
rfr_drop_mse_scores
```

```
array([9.89141646e+08, 8.15899050e+08, 8.06379018e+09, 6.23110866e+10,
       1.49205473e+09, 9.57644340e+08, 1.14373324e+10, 2.61438110e+09,
       2.83141180e+10, 1.60552968e+09])
```

```
#Create function to compute % similarity of the prediction vs actual y value.
# % similarity = 1 - % difference
keras.utils.get_custom_objects().clear()
@keras.utils.register_keras_serializable(package="my_package", name="percent_sim")
def percent_sim(y_true, y_pred):
  mean = (y_true + y_pred)/2
  diff = abs(y_true - y_pred)
  percent_diff = diff/mean
  return 1 - percent_diff
```

```
#Create callback to save the weights after each epoch
class customCallback(callbacks.Callback):
  def on_train_begin(self,logs=None):
    self.model.all_weights = []
  def on_epoch_end(self,epoch,logs=None):
    self.weight_list = []
    for layer in self.model.layers:
      if layer.get_weights() == []:
        self.weight_list += [np.nan]
      else:
        self.weight_list += [np.mean(layer.get_weights()[0])]
    self.model.all_weights += [self.weight_list]
```

Create and train initial Convolutional Neural Network

```
def build_cnn(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    model.add(Dense(50, activation='relu')) # Adds dense layer (fully connected), 50 neurons and again uses relu activation
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    model.compile(optimizer='adam', loss='mse', metrics=['mse',percent_sim]) # Compiles the model, uses Adam optimizer which combines the ad
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 12 features, 1 row per input
cnn_model = build_cnn(cnn_input_shape)
cnn_model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_318"

Layer (type)	Output Shape	Param #
conv1d_318 (Conv1D)	(None, 10, 64)	256
max_pooling1d_318 (MaxPooling1D)	(None, 5, 64)	0
flatten_318 (Flatten)	(None, 320)	0
dense_1037 (Dense)	(None, 50)	16,050
dense_1038 (Dense)	(None, 1)	51

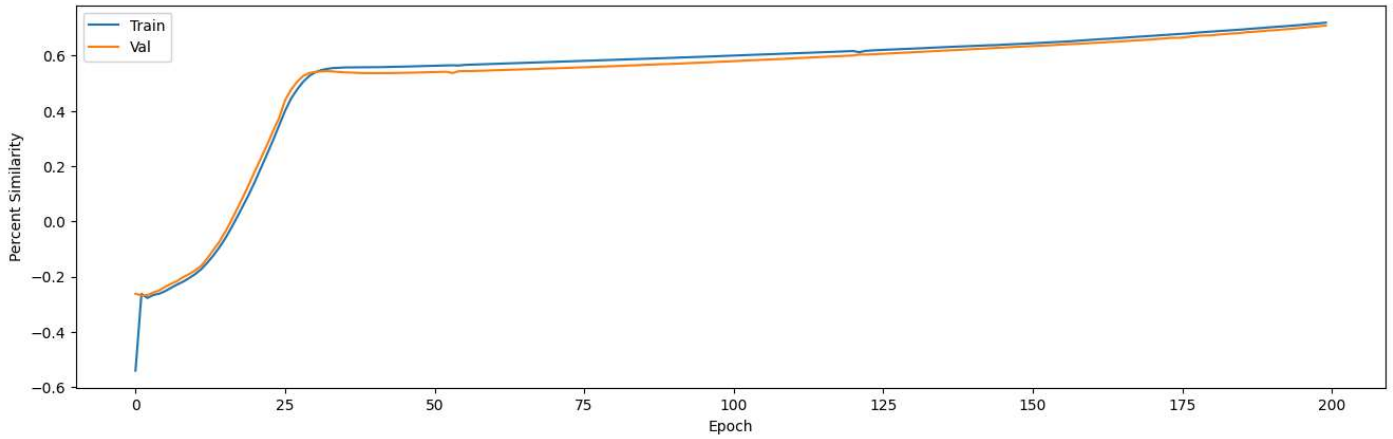
Total params: 16,357 (63.89 KB)
Trainable params: 16,357 (63.89 KB)
Non-trainable params: 0 (0.00 B)

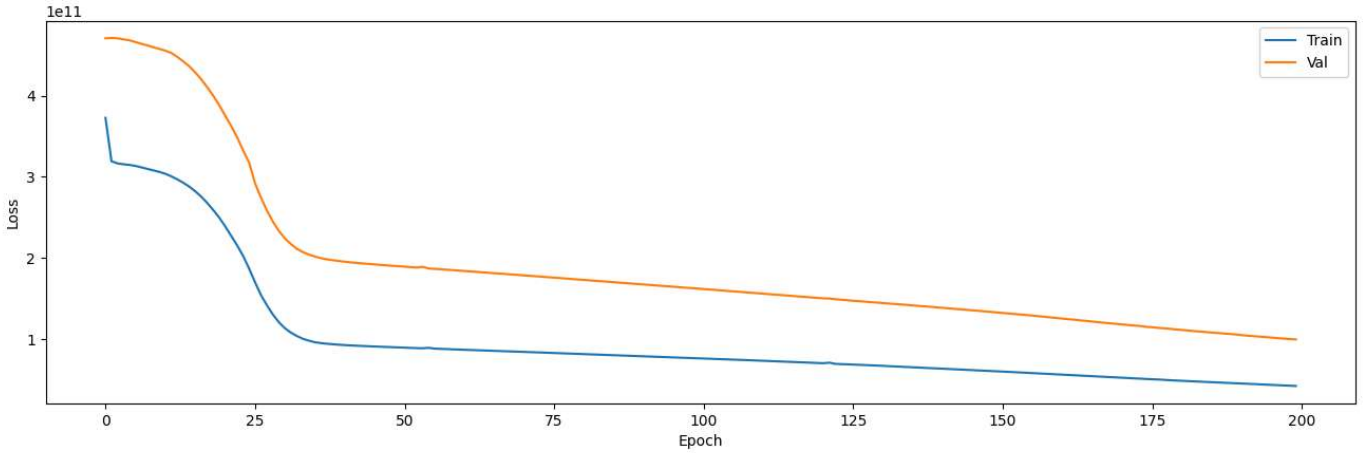
```
print(X_train.shape)
print(y_train.shape)
```

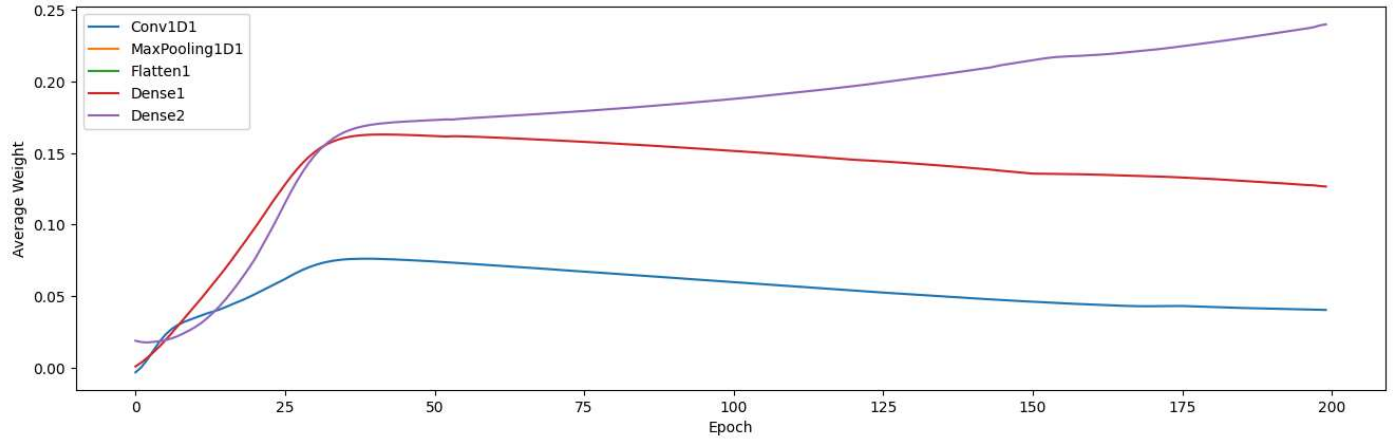
```
(585, 12)
(585,)
```

```
cnn_history = cnn_model.fit(X_train, y_train, epochs=200, batch_size=32, validation_data=(X_val,y_val), callbacks=[customCallback()], verbos


cnn_df = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df['epoch'] = cnn_history.epoch
cnn_df['train_percent'] = cnn_history.history['percent_sim']
cnn_df['val_percent'] = cnn_history.history['val_percent_sim']
cnn_df['loss'] = cnn_history.history['loss']
cnn_df['val_loss'] = cnn_history.history['val_loss']

name = []
for layer in cnn_model.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model.all_weights):
  model_layers.loc[i] = w



plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df,label="Train")
plt.plot('epoch','val_percent',data=cnn_df,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df,label="Train")
plt.plot('epoch','val_loss',data=cnn_df,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers.columns:
  plt.plot(model_layers.index,col,data=model_layers,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1])
plt.legend()

plt.show()
```

[4]







Perform grid search on various hyperparameters for the Adam optimizer.

```
def build_cnn_new(learning_rate=0.05,beta_1=0.9,beta_2=0.999):
    input_shape = (12,1)
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    model.add(Dense(50, activation='relu')) # Adds dense layer (fully connected), 50 neurons and again uses relu activation
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    opt = tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=beta_1, beta_2=beta_2)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model


learning_rate = [0.0005,0.001,0.005,0.01,0.05]
beta_1 = [0.9,0.99,0.999]
beta_2 = [0.9,0.99,0.999]
param_grid = dict(model__optimizer__learning_rate=learning_rate, model__optimizer__beta_1=beta_1, model__optimizer__beta_2=beta_2)
estimator = KerasRegressor(model=build_cnn_new, epochs=50, batch_size=32, validation_split=0.2, callbacks=[customCallback()])
grid1 = GridSearchCV(estimator=estimator, param_grid=param_grid, cv=2, scoring='neg_mean_squared_error', return_train_score=True)


grid1_history = grid1.fit(X_train,y_train,verbose=0)
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
grid1_history.best_score_
```

```
-81954160640.0
```

```
grid1_history.best_params_
```

```
{'model__optimizer__beta_1': 0.9,
 'model__optimizer__beta_2': 0.99,
 'model__optimizer__learning_rate': 0.0005}
```

Create new CNN with updated hyperparameters slightly tweaked from best results of the grid search results. (We will be slightly tweaking the Adam optimizer parameters throughout the model refining process to get the best results for each model we try)

```
def build_cnn_adam_upt(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and one

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons, ea
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alon
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    model.add(Dense(50, activation='relu')) # Adds dense layer (fully connected), 50 neurons and again uses relu activation
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. Act
    #model_weights = weights(model.layers[0])
    opt = tf.keras.optimizers.Adam(learning_rate=0.005, beta_1=0.99, beta_2=0.99)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 13 features, 1 row per input
cnn_model_adam_upt = build_cnn_adam_upt(cnn_input_shape)
cnn_model_adam_upt.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_617"

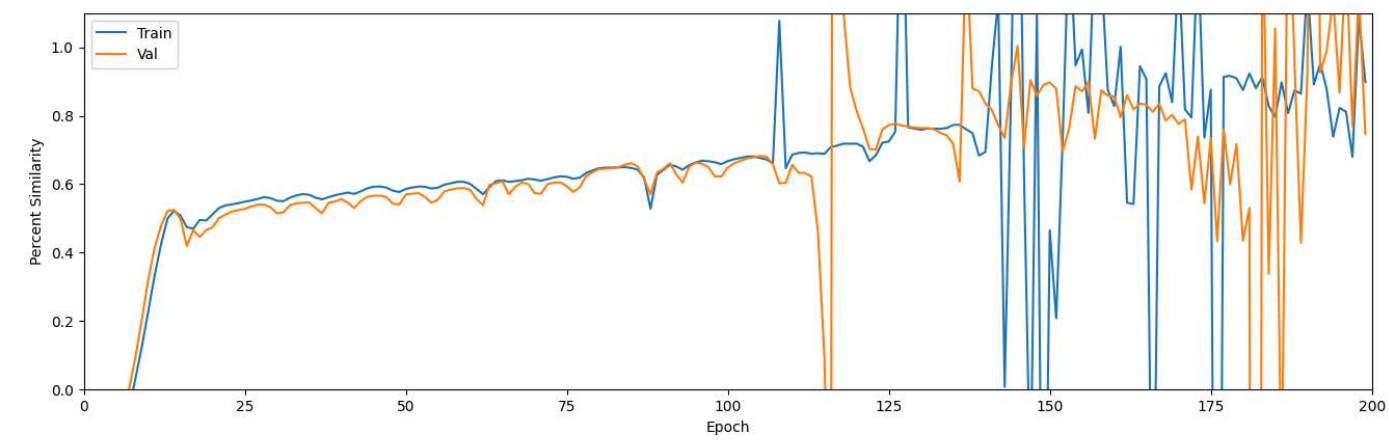
Layer (type)	Output Shape	Param #
conv1d_617 (Conv1D)	(None, 10, 64)	256
max_pooling1d_617 (MaxPooling1D)	(None, 5, 64)	0
flatten_617 (Flatten)	(None, 320)	0
dense_1764 (Dense)	(None, 50)	16,050
dense_1765 (Dense)	(None, 1)	51

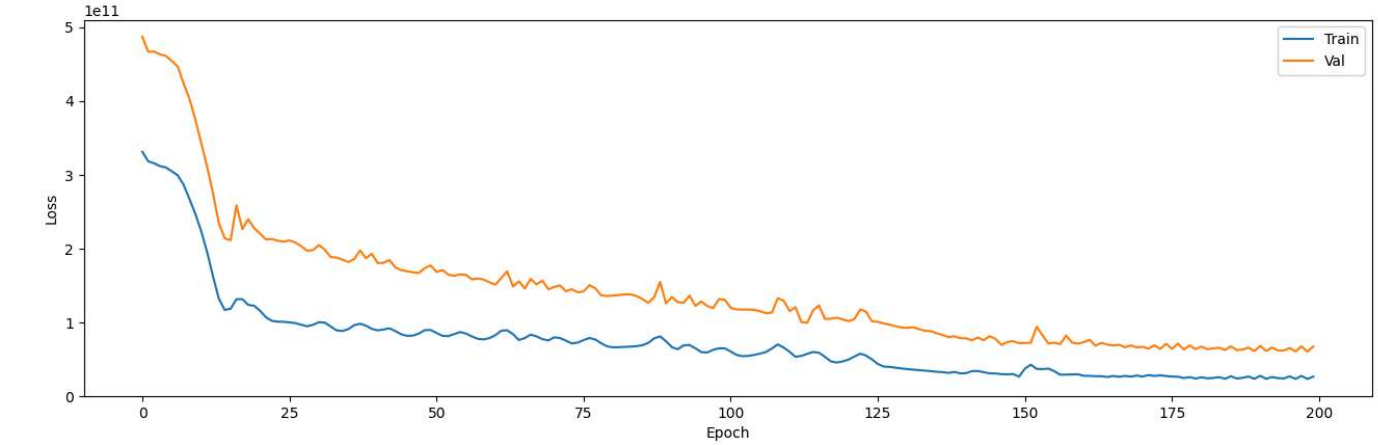
Total params: 16,357 (63.89 KB)
Trainable params: 16,357 (63.89 KB)
Non-trainable params: 0 (0.00 B)

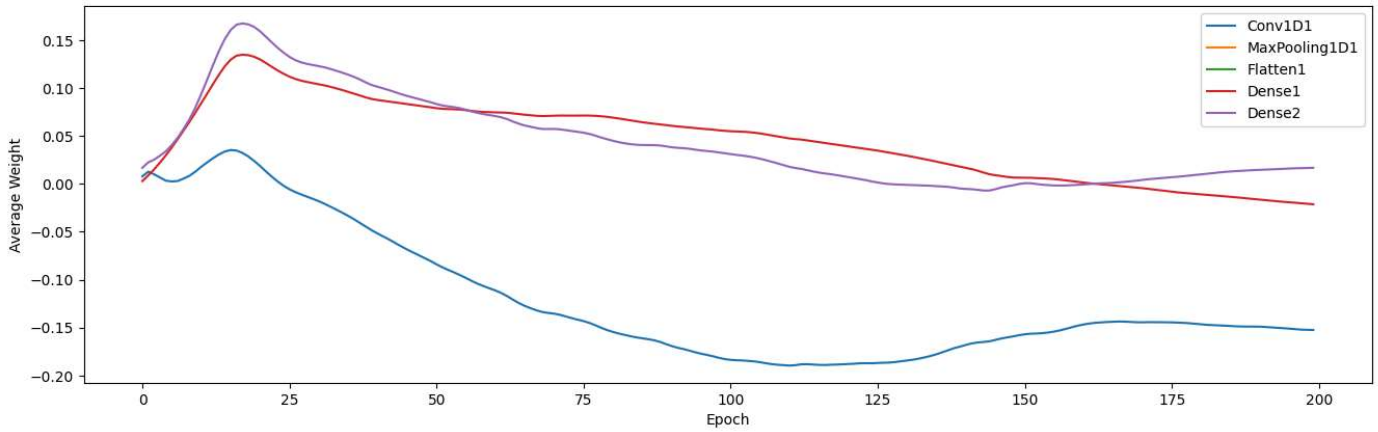
```
cnn_history_adam_upt = cnn_model_adam_upt.fit(X_train, y_train, epochs=200, batch_size=32, validation_data=(X_val,y_val), callbacks=[customC
```

```
cnn_df_adam_upt = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df_adam_upt['epoch'] = cnn_history_adam_upt.epoch
cnn_df_adam_upt['train_percent'] = cnn_history_adam_upt.history['percent_sim']
cnn_df_adam_upt['val_percent'] = cnn_history_adam_upt.history['val_percent_sim']
cnn_df_adam_upt['loss'] = cnn_history_adam_upt.history['loss']
cnn_df_adam_upt['val_loss'] = cnn_history_adam_upt.history['val_loss']

name = []
for layer in cnn_model_adam_upt.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers_adam_upt = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model_adam_upt.all_weights):
  model_layers_adam_upt.loc[i] = w


plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df_adam_upt,label="Train")
plt.plot('epoch','val_percent',data=cnn_df_adam_upt,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
plt.axis([0,200,0,1.1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df_adam_upt,label="Train")
plt.plot('epoch','val_loss',data=cnn_df_adam_upt,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,200,0,1.1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers_adam_upt.columns:
  plt.plot(model_layers_adam_upt.index,col,data=model_layers_adam_upt,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1.1])
plt.legend()

plt.show()
```

[4]







Above epoch vs percent similarity graph shows that the model converges more quickly and likely would need fewer epochs.

```
#Function to add given # of layers with given # of neurons
@keras.utils.register_keras_serializable(package="my_package", name="add_layer")
def add_layer(model,num_layer,neurons):
  for i in range(num_layer):
    model.add(Dense(neurons,activation='relu'))
```

Perform grid search on various numbers of layers to add and various numbers of neurons per layer

```
def build_cnn_add_layers(num_layer=2, neurons=50):
    input_shape = (12,1)
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer(model,num_layer,neurons)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    opt = tf.keras.optimizers.Adam(learning_rate=0.005, beta_1=0.9, beta_2=0.999)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model


num_layers = [1,2,5,10,20]
neurons = [100,50,20]
param_grid2 = dict(model__add_layer__num_layer=num_layers, model__add_layer__neurons=neurons)
estimator2 = KerasRegressor(model=build_cnn_add_layers, epochs=100, batch_size=32, validation_split=0.2, callbacks=[customCallback()])
grid2 = GridSearchCV(estimator=estimator2, param_grid=param_grid2, cv=2, scoring='neg_mean_squared_error', return_train_score=True)


grid2_history = grid2.fit(X_train,y_train,verbose=0)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

grid2_history.cv_results_


```
                    20, 20, 20],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value=999999),
 'param_model__add_layer__num_layer': masked_array(data=[1, 2, 5, 10, 20, 1, 2, 5, 10, 20, 1, 2, 5, 10, 20],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value=999999),
 'params': [{'model__add_layer__neurons': 100,
   'model__add_layer__num_layer': 1},
  {'model__add_layer__neurons': 100, 'model__add_layer__num_layer': 2},
  {'model__add_layer__neurons': 100, 'model__add_layer__num_layer': 5},
  {'model__add_layer__neurons': 100, 'model__add_layer__num_layer': 10},
  {'model__add_layer__neurons': 100, 'model__add_layer__num_layer': 20},
  {'model__add_layer__neurons': 50, 'model__add_layer__num_layer': 1},
  {'model__add_layer__neurons': 50, 'model__add_layer__num_layer': 2},
  {'model__add_layer__neurons': 50, 'model__add_layer__num_layer': 5},
  {'model__add_layer__neurons': 50, 'model__add_layer__num_layer': 10},
  {'model__add_layer__neurons': 50, 'model__add_layer__num_layer': 20},
  {'model__add_layer__neurons': 20, 'model__add_layer__num_layer': 1},
  {'model__add_layer__neurons': 20, 'model__add_layer__num_layer': 2},
  {'model__add_layer__neurons': 20, 'model__add_layer__num_layer': 5},
  {'model__add_layer__neurons': 20, 'model__add_layer__num_layer': 10},
  {'model__add_layer__neurons': 20, 'model__add_layer__num_layer': 20}],
 'split0_test_score': array([-7.63685356e+10, -7.48712591e+10, -7.12328806e+10, -7.84586506e+10,
        -7.17484360e+10, -7.26363750e+10, -7.73340037e+10, -7.60642601e+10,
        -8.35154084e+10, -7.16776489e+10, -8.03134751e+10, -8.44637635e+10,
        -7.80007752e+10, -7.58791700e+10, -7.66057267e+10]),
 'split1_test_score': array([-7.43311933e+10, -7.78527785e+10, -7.80612731e+10, -8.08260895e+10,
        -7.08994171e+10, -8.13892731e+10, -7.90439772e+10, -7.20774349e+10,
        -7.38703196e+10, -8.24546591e+10, -8.07162675e+10, -7.17950894e+10,
        -8.27399782e+10, -7.04317440e+10, -6.67508204e+10]),
 'mean_test_score': array([-7.53498644e+10, -7.63620188e+10, -7.46470769e+10, -7.96423700e+10,
        -7.13239265e+10, -7.70128241e+10, -7.81889905e+10, -7.40708475e+10,
        -7.86928640e+10, -7.70661540e+10, -8.05148713e+10, -7.81294264e+10,
        -8.03703767e+10, -7.31554570e+10, -7.16782735e+10]),
 'std_test_score': array([1.01867110e+09, 1.49075968e+09, 3.41419622e+09, 1.18371942e+09,
        4.24509440e+08, 4.37644902e+09, 8.54986752e+08, 1.99341261e+09,
        4.82254438e+09, 5.38850509e+09, 2.01396224e+08, 6.33433702e+09,
        2.36960154e+09, 2.72371302e+09, 4.92745318e+09]),
 'rank_test_score': array([ 6,  7,  5, 13,  1,  8, 11,  4, 12,  9, 15, 10, 14,  3,  2],
       dtype=int32),
 'split0_train_score': array([-6.03845919e+10, -5.93243955e+10, -5.08474368e+10, -5.95497820e+10,
        -4.58405806e+10, -5.67105577e+10, -5.94664489e+10, -5.77241252e+10,
        -5.46896445e+10, -4.90891796e+10, -5.82004490e+10, -6.67587420e+10,
        -5.54649231e+10, -4.82886697e+10, -6.33040323e+10]),
 'split1_train_score': array([-7.39291136e+10, -7.97330227e+10, -8.00120750e+10, -8.23598940e+10,
        -7.12561132e+10, -7.76380744e+10, -7.47342479e+10, -6.91867402e+10,
        -7.23619676e+10, -8.41067889e+10, -7.89689958e+10, -7.30406011e+10,
        -8.19579208e+10, -7.47668439e+10, -6.69380567e+10]),
 'mean_train_score': array([-6.71568527e+10, -6.95287091e+10, -6.54297559e+10, -7.09548380e+10,
        -5.85483469e+10, -6.71743160e+10, -6.71003484e+10, -6.34554327e+10,
        -6.35258061e+10, -6.65979843e+10, -6.85847224e+10, -6.98996716e+10,
        -6.87114220e+10, -6.15277568e+10, -6.51210445e+10]),
 'std_train_score': array([6.77226086e+09, 1.02043136e+10, 1.45823191e+10, 1.14050560e+10,
        1.27077663e+10, 1.04637583e+10, 7.63389952e+09, 5.73130752e+09,
        8.83616154e+09, 1.75088046e+10, 1.03842734e+10, 3.14092954e+09,
        1.32464988e+10, 1.32390871e+10, 1.81701222e+09])}
```


grid2_history.best_score_

```
-71323926528.0
```

grid2_history.best_params_

```
{'model__add_layer__neurons': 100, 'model__add_layer__num_layer': 20}
```

Update model to add an optimal 20 layers with 100 neurons each.

```
def build_cnn_add_layers_upt(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer(model,20,100)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    #model_weights = weights(model.layers[0])
    opt = tf.keras.optimizers.Adam(learning_rate=0.005, beta_1=0.9, beta_2=0.999)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    #model.compile(optimizer='adam', loss='mse', metrics=['mse',percent_sim,model_weights]) # Compiles the model, uses Adam optimizer which
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 13 features, 1 row per input
cnn_model_add_layers_upt = build_cnn_add_layers_upt(cnn_input_shape)
cnn_model_add_layers_upt.summary()
```

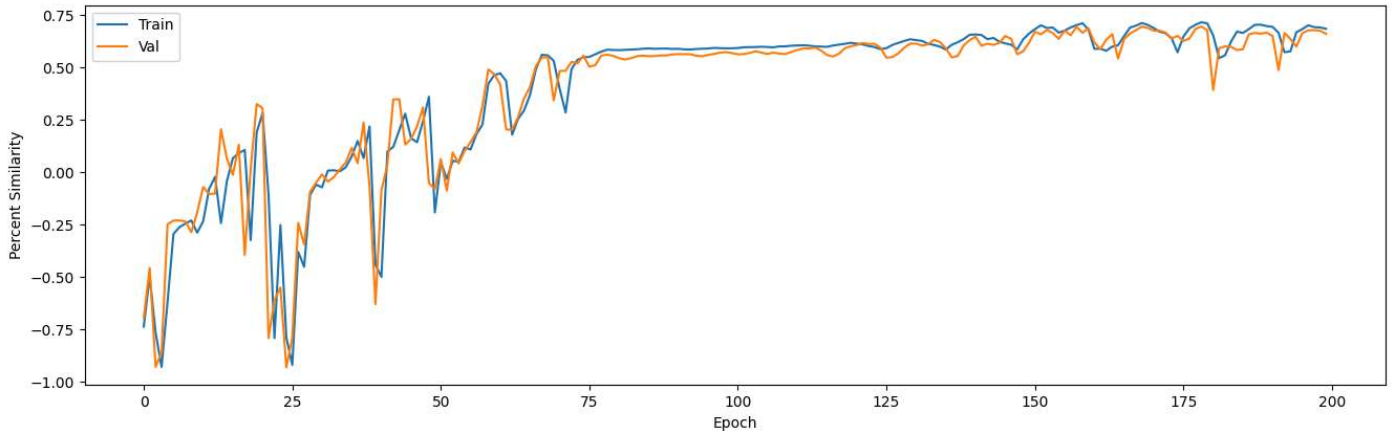
Model: "sequential_744"

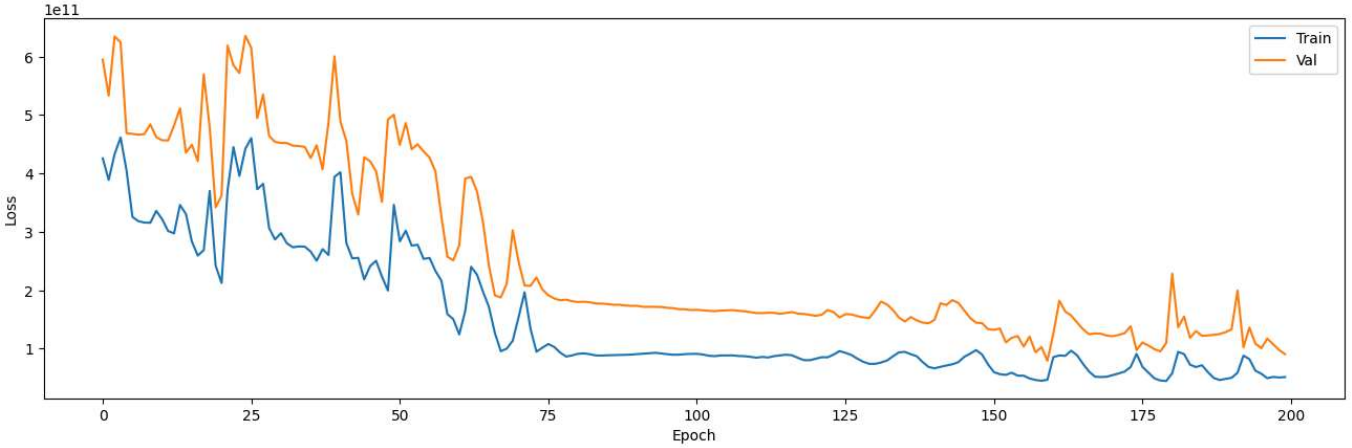
Layer (type)	Output Shape	Param #
conv1d_744 (Conv1D)	(None, 10, 64)	256
max_pooling1d_744 (MaxPooling1D)	(None, 5, 64)	0
flatten_744 (Flatten)	(None, 320)	0
dense_2160 (Dense)	(None, 100)	32,100
dense_2161 (Dense)	(None, 100)	10,100
dense_2162 (Dense)	(None, 100)	10,100
dense_2163 (Dense)	(None, 100)	10,100
dense_2164 (Dense)	(None, 100)	10,100
dense_2165 (Dense)	(None, 100)	10,100
dense_2166 (Dense)	(None, 100)	10,100
dense_2167 (Dense)	(None, 100)	10,100
dense_2168 (Dense)	(None, 100)	10,100
dense_2169 (Dense)	(None, 100)	10,100
dense_2170 (Dense)	(None, 100)	10,100
dense_2171 (Dense)	(None, 100)	10,100
dense_2172 (Dense)	(None, 100)	10,100
dense_2173 (Dense)	(None, 100)	10,100
dense_2174 (Dense)	(None, 100)	10,100
dense_2175 (Dense)	(None, 100)	10,100
dense_2176 (Dense)	(None, 100)	10,100
dense_2177 (Dense)	(None, 100)	10,100
dense_2178 (Dense)	(None, 100)	10,100
dense_2179 (Dense)	(None, 100)	10,100
dense_2180 (Dense)	(None, 1)	101

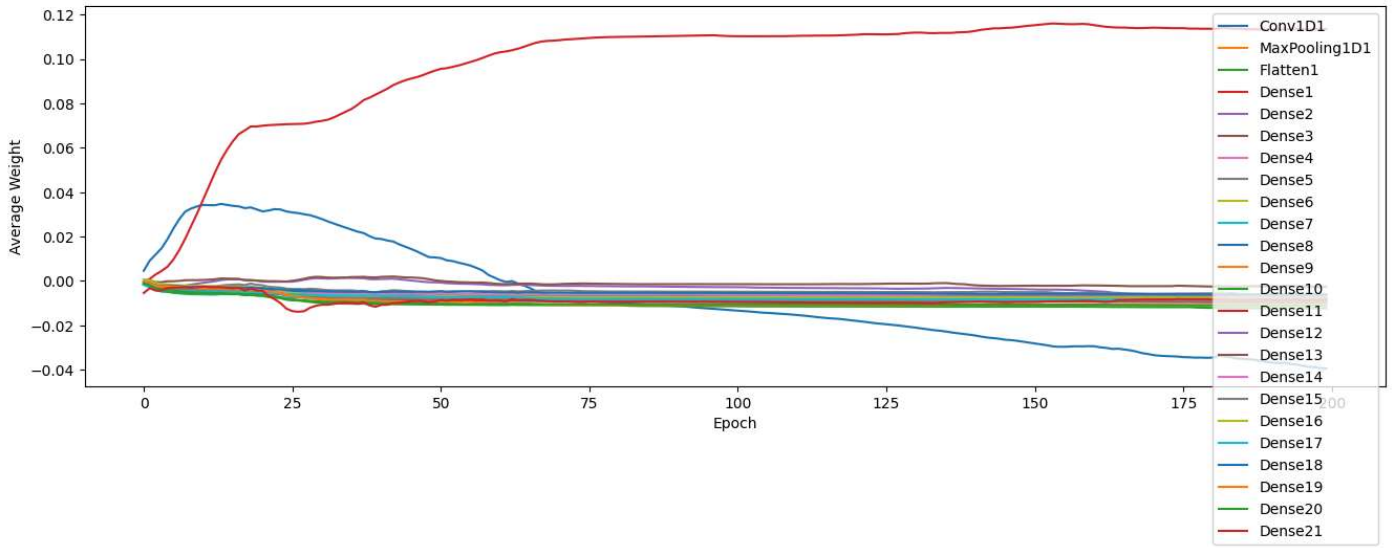
```
cnn_history_add_layers_upt = cnn_model_add_layers_upt.fit(X_train, y_train, epochs=200, batch_size=128, validation_data=(X_val,y_val), callb
```

```
cnn_df_add_layers_upt = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df_add_layers_upt['epoch'] = cnn_history_add_layers_upt.epoch
cnn_df_add_layers_upt['train_percent'] = cnn_history_add_layers_upt.history['percent_sim']
cnn_df_add_layers_upt['val_percent'] = cnn_history_add_layers_upt.history['val_percent_sim']
cnn_df_add_layers_upt['loss'] = cnn_history_add_layers_upt.history['loss']
cnn_df_add_layers_upt['val_loss'] = cnn_history_add_layers_upt.history['val_loss']

name = []
for layer in cnn_model_add_layers_upt.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers_add_layers_upt = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model_add_layers_upt.all_weights):
  model_layers_add_layers_upt.loc[i] = w


plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df_add_layers_upt,label="Train")
plt.plot('epoch','val_percent',data=cnn_df_add_layers_upt,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df_add_layers_upt,label="Train")
plt.plot('epoch','val_loss',data=cnn_df_add_layers_upt,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers_add_layers_upt.columns:
  plt.plot(model_layers_add_layers_upt.index,col,data=model_layers_add_layers_upt,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1])
plt.legend()

plt.show()
```

[4]


Train
Val
Percent Similarity
Epoch


1e11
Train
Val
Loss
Epoch


Conv1D1
MaxPooling1D1
Flatten1
Dense1
Dense2
Dense3
Dense4
Dense5
Dense6
Dense7
Dense8
Dense9
Dense10
Dense11
Dense12
Dense13
Dense14
Dense15
Dense16
Dense17
Dense18
Dense19
Dense20
Dense21
Average Weight
Epoch

Percent similarity has a great deal of variance in the earlier epochs and flattens out at a rate of less than 0.75. We will now decrease the number of layers added to 10, and decrease the neurons per layer to 50, and see if this helps improve the graphs.

```
def build_cnn_add_layers_10(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and one

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons, ea
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alon
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer(model,10,50)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. Act
    opt = tf.keras.optimizers.Adam(learning_rate=0.005, beta_1=0.99, beta_2=0.99)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 13 features, 1 row per input
cnn_model_add_layers_10 = build_cnn_add_layers_10(cnn_input_shape)
cnn_model_add_layers_10.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_712"

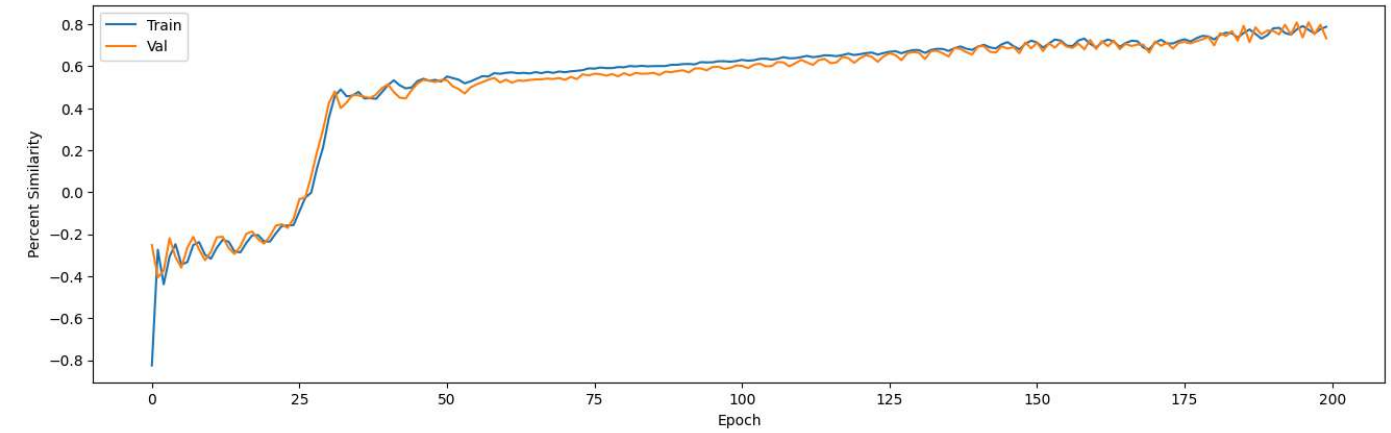
Layer (type)	Output Shape	Param #
conv1d_712 (Conv1D)	(None, 10, 64)	256
max_pooling1d_712 (MaxPooling1D)	(None, 5, 64)	0
flatten_712 (Flatten)	(None, 320)	0
dense_2056 (Dense)	(None, 50)	16,050
dense_2057 (Dense)	(None, 50)	2,550
dense_2058 (Dense)	(None, 50)	2,550
dense_2059 (Dense)	(None, 50)	2,550
dense_2060 (Dense)	(None, 50)	2,550
dense_2061 (Dense)	(None, 50)	2,550
dense_2062 (Dense)	(None, 50)	2,550
dense_2063 (Dense)	(None, 50)	2,550
dense_2064 (Dense)	(None, 50)	2,550
dense_2065 (Dense)	(None, 50)	2,550
dense_2066 (Dense)	(None, 1)	51

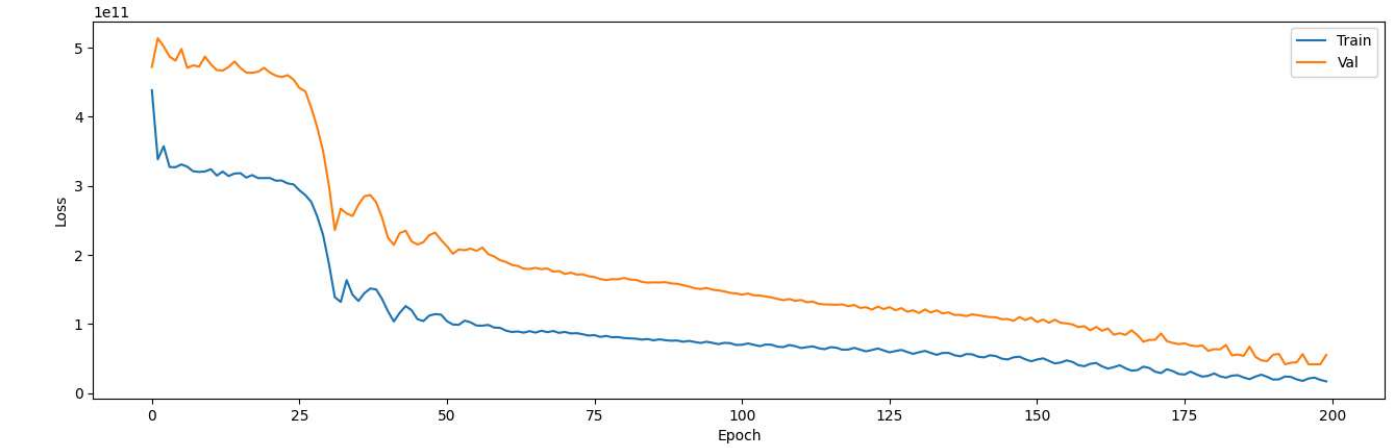
Total params: 39,307 (153.54 KB)

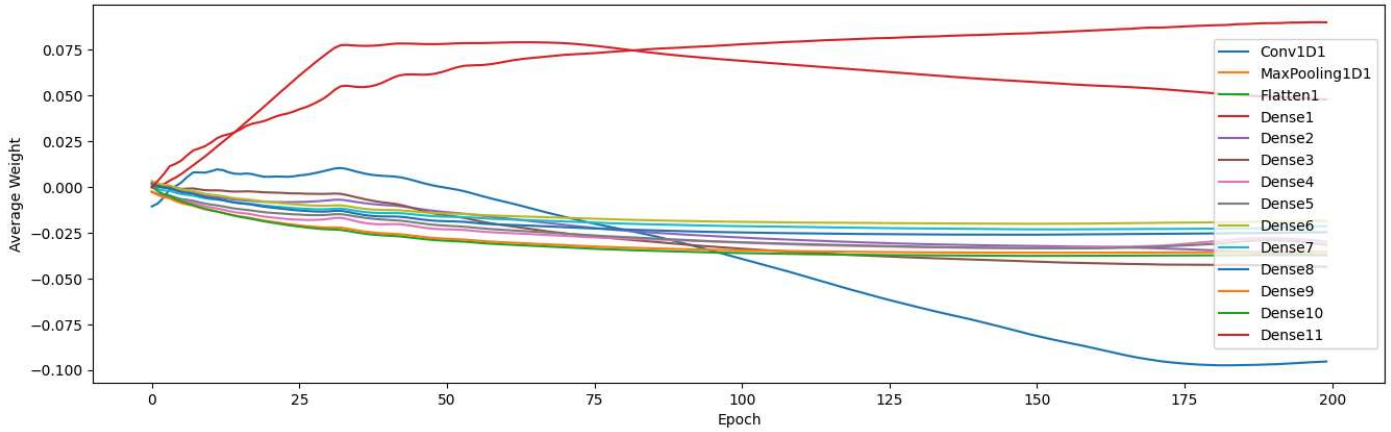
```
cnn_history_add_layers_10 = cnn_model_add_layers_10.fit(X_train, y_train, epochs=200, batch_size=128, validation_data=(X_val,y_val), callbac
```

```
cnn_df_add_layers_10 = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df_add_layers_10['epoch'] = cnn_history_add_layers_10.epoch
cnn_df_add_layers_10['train_percent'] = cnn_history_add_layers_10.history['percent_sim']
cnn_df_add_layers_10['val_percent'] = cnn_history_add_layers_10.history['val_percent_sim']
cnn_df_add_layers_10['loss'] = cnn_history_add_layers_10.history['loss']
cnn_df_add_layers_10['val_loss'] = cnn_history_add_layers_10.history['val_loss']

name = []
for layer in cnn_model_add_layers_10.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers_add_layers_10 = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model_add_layers_10.all_weights):
  model_layers_add_layers_10.loc[i] = w


plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df_add_layers_10,label="Train")
plt.plot('epoch','val_percent',data=cnn_df_add_layers_10,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df_add_layers_10,label="Train")
plt.plot('epoch','val_loss',data=cnn_df_add_layers_10,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers_add_layers_10.columns:
  plt.plot(model_layers_add_layers_10.index,col,data=model_layers_add_layers_10,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1])
plt.legend()

plt.show()
```

[4]







Tweaking the parameters slightly has inproved the graphs a lot, and the percent similarity is now around 0.8.

```
#Function to add layers with Leaky ReLU activation with given slope
@keras.utils.register_keras_serializable(package="my_package", name="add_layer_lr")
def add_layer_lr(model,num_layer,neurons,slope):
  act = tf.keras.layers.LeakyReLU(negative_slope=slope)
  for i in range(num_layer):
    model.add(Dense(neurons,activation=act))
```

Create CNN with Leaky ReLU, and perform a grid search on multiple slopes

```
def build_cnn_leaky_relu(slope=0.1):
    input_shape = (12,1)
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer_lr(model,10,50,slope)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    opt = tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.99, beta_2=0.999)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model
```

```
slope = [0.1,0.3,0.5,0.8]
param_grid3 = dict(model__add_layer__slope=slope)
estimator3 = KerasRegressor(model=build_cnn_leaky_relu, epochs=100, batch_size=32, validation_split=0.2, callbacks=[customCallback()])
grid3 = GridSearchCV(estimator=estimator3, param_grid=param_grid3, cv=3, scoring='neg_mean_squared_error', return_train_score=True)
```

```
grid3_history = grid3.fit(X_train,y_train,verbose=0)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
grid3_history.cv_results_
```

```
{'mean_fit_time': array([15.35448122, 15.36425241, 15.69075886, 15.51357214]),
 'std_fit_time': array([0.23921189, 0.25807223, 0.51680006, 0.190671  ]),
 'mean_score_time': array([0.45985778, 0.35108415, 0.39575521, 0.4496909 ]),
 'std_score_time': array([0.1792878 , 0.02104211, 0.07326971, 0.18825555]),
 'param_model__add_layer__slope': masked_array(data=[0.1, 0.3, 0.5, 0.8],
             mask=[False, False, False, False],
       fill_value=1e+20),
 'params': [{'model__add_layer__slope': 0.1},
  {'model__add_layer__slope': 0.3},
  {'model__add_layer__slope': 0.5},
  {'model__add_layer__slope': 0.8}],
 'split0_test_score': array([-1.14865299e+11, -5.79482337e+10, -7.02052434e+10, -6.28003963e+10]),
 'split1_test_score': array([-5.78418401e+10, -1.36852783e+11, -1.10368735e+11, -1.05593348e+11]),
 'split2_test_score': array([-5.53647432e+10, -1.13506083e+11, -1.01181645e+11, -4.71138263e+10]),
 'mean_test_score': array([-7.60239609e+10, -1.02769033e+11, -9.39185411e+10, -7.18358569e+10]),
 'std_test_score': array([2.74835852e+10, 3.30952681e+10, 1.71821812e+10, 2.47142776e+10]),
 'rank_test_score': array([2, 4, 3, 1], dtype=int32),
 'split0_train_score': array([-7.50073774e+10, -5.40822036e+10, -5.95175916e+10, -5.37644483e+10]),
 'split1_train_score': array([-4.29271941e+10, -8.00378307e+10, -9.45137828e+10, -7.52560210e+10]),
 'split2_train_score': array([-2.78594007e+10, -9.32274586e+10, -9.78968084e+10, -3.09447311e+10]),
 'mean_train_score': array([-4.85979907e+10, -7.57824976e+10, -8.39760609e+10, -5.33217335e+10]),
 'std_train_score': array([1.96613221e+10, 1.62617883e+10, 1.73498080e+10, 1.80927168e+10])}
```

```
grid3_history.best_score_
```

```
-71835856896.0
```

```
grid3_history.best_params_
```

```
{'model__add_layer__slope': 0.8}
```

Update model with the best performing parameter (0.8) for the slope

```
def build_cnn_leaky_relu(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer_lr(model,10,50,0.8)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    opt = tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.99, beta_2=0.999)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 13 features, 1 row per input
cnn_model_leaky_relu = build_cnn_leaky_relu(cnn_input_shape)
cnn_model_leaky_relu.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_775"

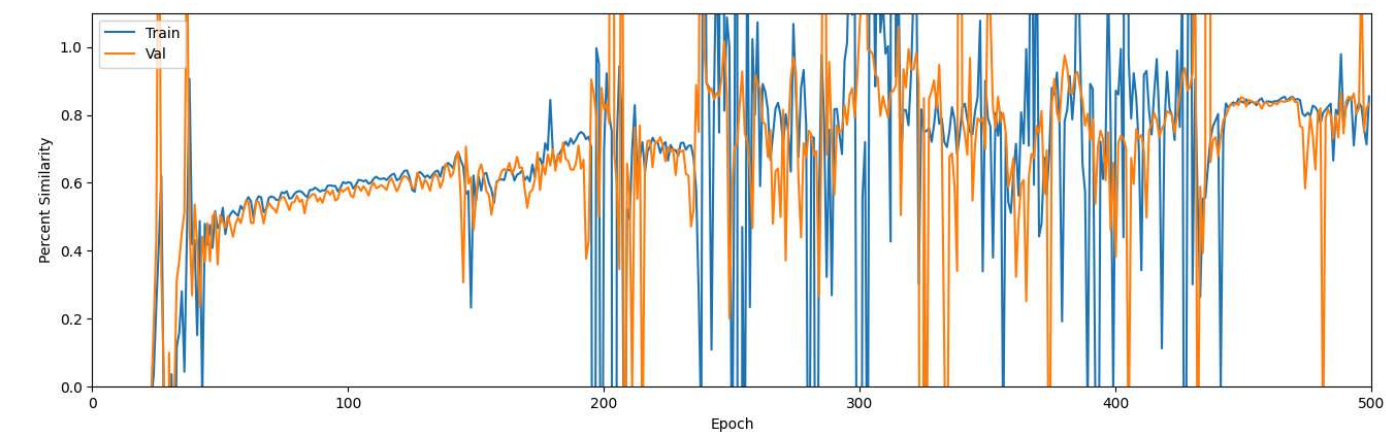
Layer (type)	Output Shape	Param #
conv1d_775 (Conv1D)	(None, 10, 64)	256
max_pooling1d_775 (MaxPooling1D)	(None, 5, 64)	0
flatten_775 (Flatten)	(None, 320)	0
dense_2511 (Dense)	(None, 50)	16,050
dense_2512 (Dense)	(None, 50)	2,550
dense_2513 (Dense)	(None, 50)	2,550
dense_2514 (Dense)	(None, 50)	2,550
dense_2515 (Dense)	(None, 50)	2,550
dense_2516 (Dense)	(None, 50)	2,550
dense_2517 (Dense)	(None, 50)	2,550
dense_2518 (Dense)	(None, 50)	2,550
dense_2519 (Dense)	(None, 50)	2,550
dense_2520 (Dense)	(None, 50)	2,550
dense_2521 (Dense)	(None, 1)	51

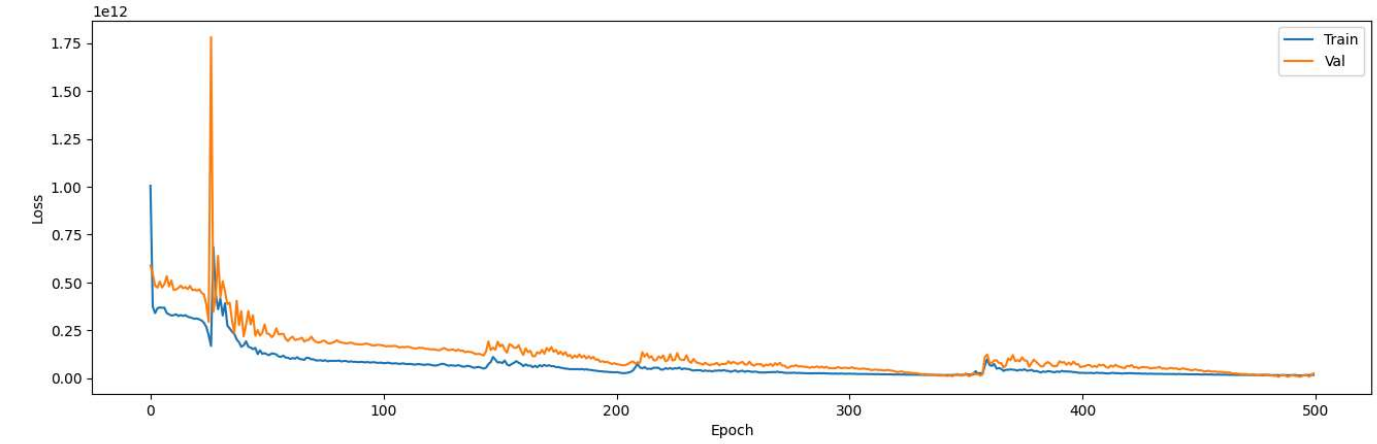
Total params: 39,307 (153.54 KB)

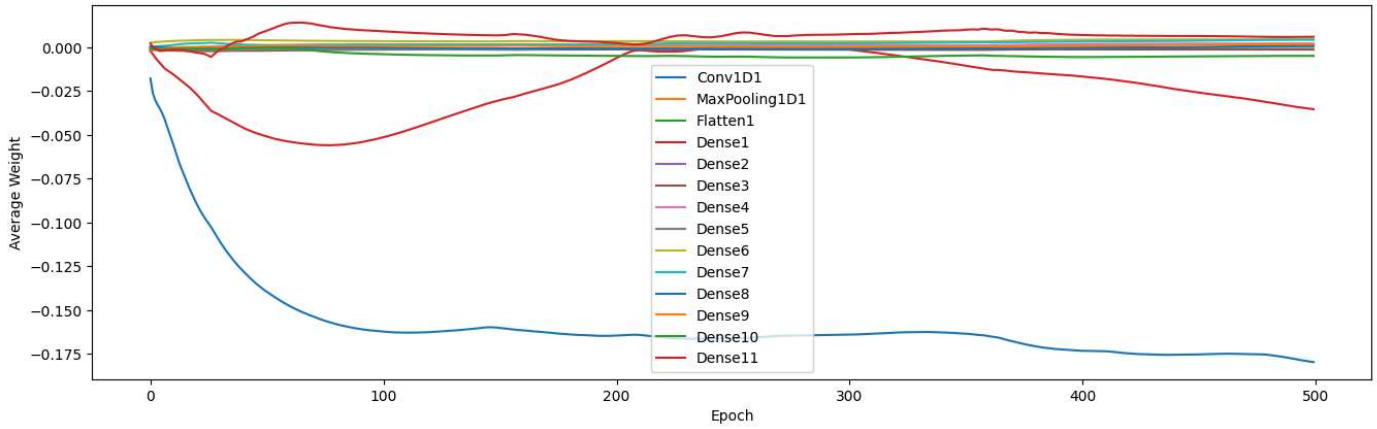
```
cnn_history_leaky_relu = cnn_model_leaky_relu.fit(X_train, y_train, epochs=500, batch_size=128, validation_data=(X_val,y_val), callbacks=[cu


cnn_df_leaky_relu = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df_leaky_relu['epoch'] = cnn_history_leaky_relu.epoch
cnn_df_leaky_relu['train_percent'] = cnn_history_leaky_relu.history['percent_sim']
cnn_df_leaky_relu['val_percent'] = cnn_history_leaky_relu.history['val_percent_sim']
cnn_df_leaky_relu['loss'] = cnn_history_leaky_relu.history['loss']
cnn_df_leaky_relu['val_loss'] = cnn_history_leaky_relu.history['val_loss']

name = []
for layer in cnn_model_leaky_relu.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers_leaky_relu = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model_leaky_relu.all_weights):
  model_layers_leaky_relu.loc[i] = w
```

```
plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df_leaky_relu,label="Train")
plt.plot('epoch','val_percent',data=cnn_df_leaky_relu,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
plt.axis([0,500,0,1.1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df_leaky_relu,label="Train")
plt.plot('epoch','val_loss',data=cnn_df_leaky_relu,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers_leaky_relu.columns:
  plt.plot(model_layers_leaky_relu.index,col,data=model_layers_leaky_relu,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1])
plt.legend()

plt.show()
```

[4]







The graphs are a bit all over the place. While the percent similarity is heading towards upwards of 0.8, there are too many spikes and drops for this value to be reliable. We will try decreasing the slope to see if it helps

```
def build_cnn_leaky_relu_1(input_shape):
    model = Sequential() # Creates instance of Sequential model, model for NN where layers are added up in sequence (One input tensor and on

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape)) # 1D convolutional layer to model, 64 neurons,
    # Kernel size 3, each filter covers 3 units in the input dimension. Relu, allows the model to capture non-linear relationships
    model.add(MaxPooling1D(pool_size=2)) # Layer reduces the dimensionality of the data by taking the max value over a window of 2 units alo
    # Makes the model more efficient and less sensitive to small variation in the position of features
    model.add(Flatten()) # Adds flatten layer, converts multidimensional output of previous layers into a 1D array for fully connected layer
    add_layer_lr(model,10,50,0.1)
    model.add(Dense(1, activation='linear'))  # Adds another dense layer, 1 unit for output layer, single neuron whose task is regression. A
    opt = tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.99, beta_2=0.999)
    model.compile(optimizer=opt, loss='mse', metrics=['mse',percent_sim])
    # Uses MSE as loss function, typical for regression task
    return model

# Example input shape, will be in form of our data
cnn_input_shape = (12, 1)  # 13 features, 1 row per input
cnn_model_leaky_relu_1 = build_cnn_leaky_relu_1(cnn_input_shape)
cnn_model_leaky_relu_1.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_776"

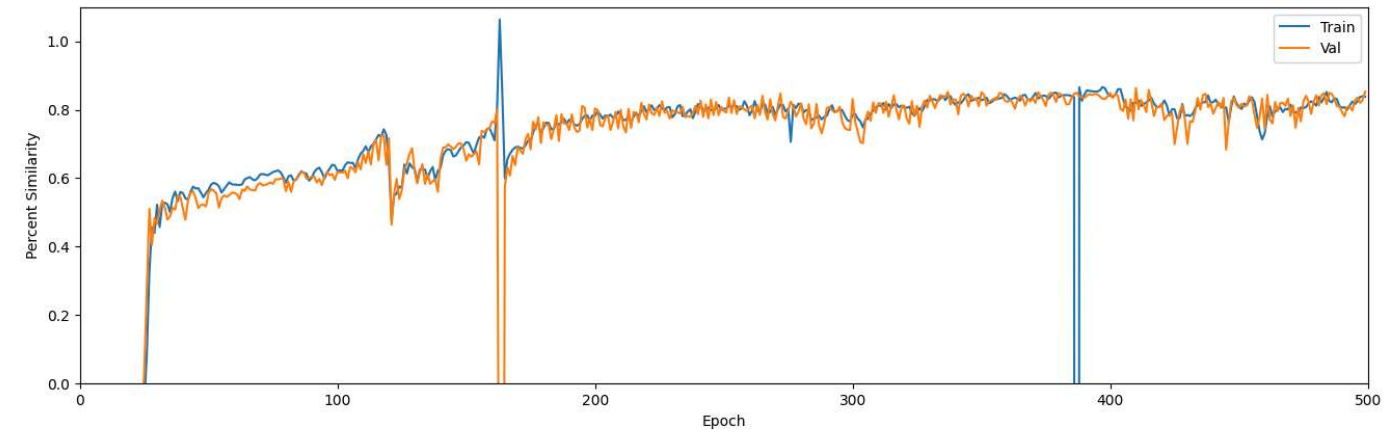
Layer (type)	Output Shape	Param #
conv1d_776 (Conv1D)	(None, 10, 64)	256
max_pooling1d_776 (MaxPooling1D)	(None, 5, 64)	0
flatten_776 (Flatten)	(None, 320)	0
dense_2522 (Dense)	(None, 50)	16,050
dense_2523 (Dense)	(None, 50)	2,550
dense_2524 (Dense)	(None, 50)	2,550
dense_2525 (Dense)	(None, 50)	2,550
dense_2526 (Dense)	(None, 50)	2,550
dense_2527 (Dense)	(None, 50)	2,550
dense_2528 (Dense)	(None, 50)	2,550
dense_2529 (Dense)	(None, 50)	2,550
dense_2530 (Dense)	(None, 50)	2,550
dense_2531 (Dense)	(None, 50)	2,550
dense_2532 (Dense)	(None, 1)	51

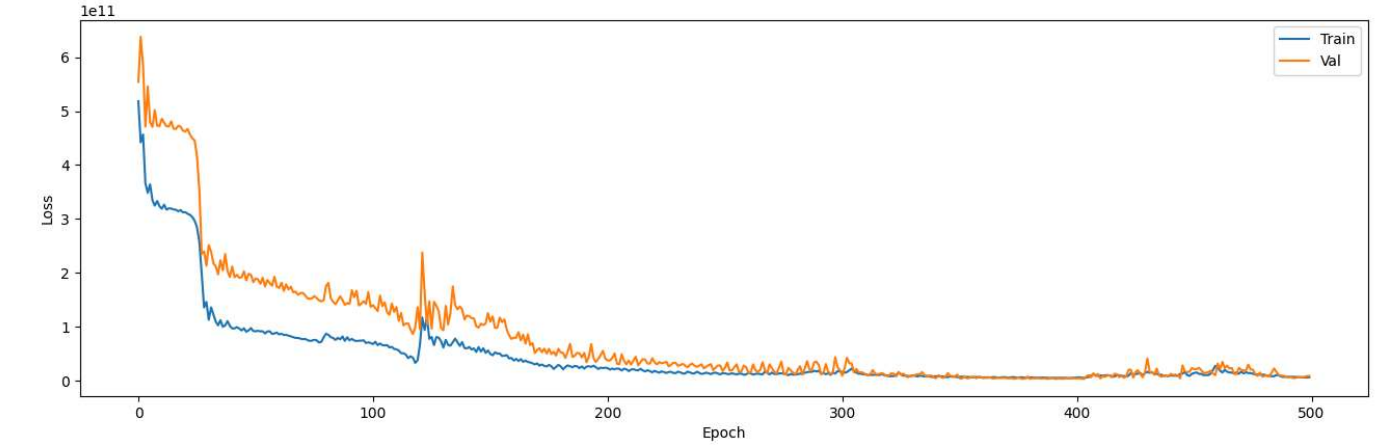
Total params: 39,307 (153.54 KB)

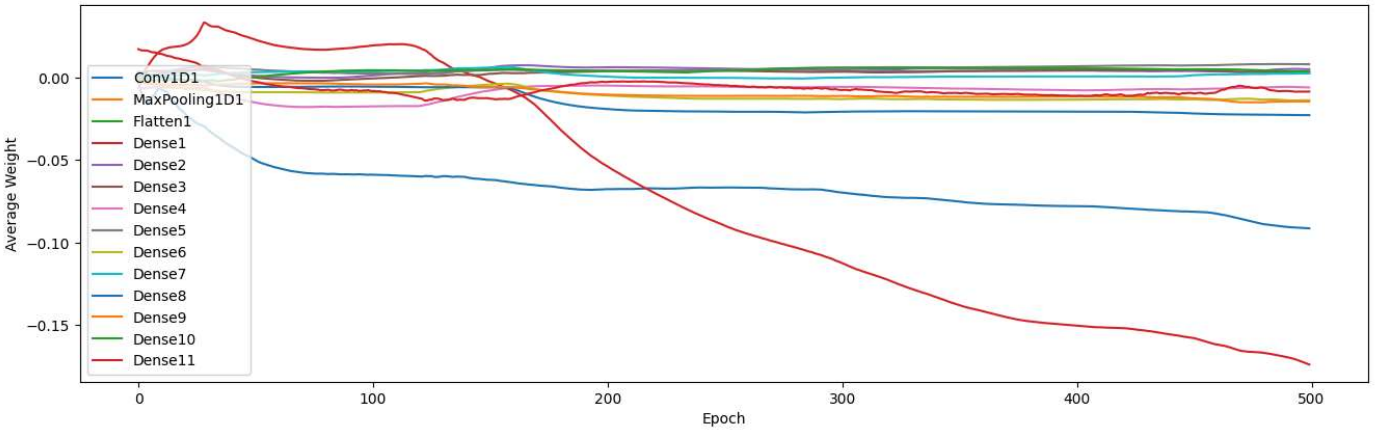
```
cnn_history_leaky_relu_1 = cnn_model_leaky_relu_1.fit(X_train, y_train, epochs=500, batch_size=128, validation_data=(X_val,y_val), callbacks


cnn_df_leaky_relu_1 = pd.DataFrame(columns=['epoch','train_percent','val_percent','loss','val_loss'])
cnn_df_leaky_relu_1['epoch'] = cnn_history_leaky_relu_1.epoch
cnn_df_leaky_relu_1['train_percent'] = cnn_history_leaky_relu_1.history['percent_sim']
cnn_df_leaky_relu_1['val_percent'] = cnn_history_leaky_relu_1.history['val_percent_sim']
cnn_df_leaky_relu_1['loss'] = cnn_history_leaky_relu_1.history['loss']
cnn_df_leaky_relu_1['val_loss'] = cnn_history_leaky_relu_1.history['val_loss']

name = []
for layer in cnn_model_leaky_relu_1.layers:
  name += [str(type(layer)).split("'")[1].split(".")[5]]
name_count = pd.Series(name).value_counts()
name_count_dict = name_count.to_dict()
for i in name_count_dict.keys():
  start = 0
  for j in range(name_count_dict[i]):
    ind = name.index(i,start)
    name[ind] = str(name[ind]) + str(j + 1)
    start = j + 1
model_layers_leaky_relu_1 = pd.DataFrame(columns=name)
for i,w in enumerate(cnn_model_leaky_relu_1.all_weights):
  model_layers_leaky_relu_1.loc[i] = w
```

```
  model_layers_leaky_relu_1[1]  ..


plt.figure(figsize=(16,16))

plt.subplot(3,1,1)
plt.plot('epoch','train_percent',data=cnn_df_leaky_relu_1,label="Train")
plt.plot('epoch','val_percent',data=cnn_df_leaky_relu_1,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Percent Similarity")
plt.axis([0,500,0,1.1])
plt.legend()

plt.subplot(3,1,2)
plt.plot('epoch','loss',data=cnn_df_leaky_relu_1,label="Train")
plt.plot('epoch','val_loss',data=cnn_df_leaky_relu_1,label='Val')
plt.xlabel("Epoch")
plt.ylabel("Loss")
#plt.axis([0,500,0,1])
plt.legend()

plt.subplot(3,1,3)
for col in model_layers_leaky_relu_1.columns:
  plt.plot(model_layers_leaky_relu_1.index,col,data=model_layers_leaky_relu_1,label=col)
plt.xlabel("Epoch")
plt.ylabel("Average Weight")
#plt.axis([0,500,0,1])
plt.legend()

plt.show()
```

[1]








```
cnn_df_leaky_relu_1.tail(10)
```