1. Set up

In [1]:
```
import os
import cv2
import matplotlib.pyplot as plt
from concurrent.futures import ThreadPoolExecutor
import insightface
from insightface.app import FaceAnalysis
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
import scipy
import onnxruntime as ort
import shutil
import random
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, random_split
from PIL import Image
import torchvision.transforms as transforms
import torch.nn.functional as F
from torch.nn import Parameter
import math
import torchvision.datasets as datasets
from torch.nn import DataParallel
from torch.optim.lr_scheduler import StepLR
import time
from sklearn.model_selection import train_test_split
from math import floor
from torchvision import datasets, transforms, models
import timm
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
import re
```

```
/home/xiaoxin/.conda/envs/face/lib/python3.8/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Pl
ease update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

2. Data loading, pre-processing, and exploration

The dataset is stored in a 100 folds which contain 400 images.

This dataset is for Age estimation: Predict the age of individuals in the images using deep learning model, such as resnet18, efficientnet and swin-tiny.

2.1 Data loading

Data ingestion function definition

In [3]:
```
# Load image
def load_image(image_path):
    img = cv2.imread(image_path, 1)
    if img is None:
        return None
    else:
        return img

# Get image paths from image folder
def get_image_paths(image_folder):
    image_paths = list()
    for root, dirs, files in os.walk(image_folder):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png')):
                image_paths.append(os.path.join(root, file))
    return image_paths

# Image path combination
def get_image_paths_from_df(image_folder, paths):
```

```
    image_paths_filter = (image_folder + paths).tolist()
    return image_paths_filter

# Display image
def image_show(images_paths):
    images = [load_image(img_path) for img_path in images_paths[0:30]]
    plt.figure(figsize=(15, 10))
    for i in range(min(30, len(images_paths))):
        plt.subplot(6, 5, i + 1)
        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
        plt.axis('off')
    plt.show()
```

Load images

In [4]:
```
image_folder = 'Wiki_Face_Small/wiki_face_small'

# Get whole paths of all images
image_paths = get_image_paths(image_folder)

# Display top 30 images
image_show(image_paths)
```

2.2 Exploration and data pre-processing (data cleaning)

1. In this part, we used insightface.app to detect images which not include any faces.
2. After cleaning the data through pretrained model, we used mat file to clean the data again.

2.2.1 Use insightface.app to filter vaild data

Data cleaning functions definition

In [5]:
```
# Detecting Faces
def detect_faces(img, app):
    if img is None or not isinstance(img, np.ndarray):
        print("Invalid image encountered.")
        return False

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    try:
```

```
        faces = app.get(img_rgb)
        return len(faces) > 0
    except Exception as e:
        print(f"Error processing image: {e}")
        return False

# Calculate ages and motify path
def calculate_ages(img_path):
    segments = img_path.replace('\\', '/').split('/')
    last_two_segments = segments[-2] + '/' + segments[-1].split('.')[0]

    # calculate age
    file_name = segments[-1]
    birth_date_str = file_name.split('_')[1]
    birth_year = int(birth_date_str.split('-')[0])

    last_year_str = file_name.split('_')[-1].split('.')[0]
    last_year = int(last_year_str)
    age = last_year - birth_year

    return age, f"{last_two_segments}.jpg"

# Check folder
def make_folder(folder_path):
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
```

Device check

In [6]:
```
# Check CUDA environment
if torch.cuda.is_available() and len(ort.get_available_providers()) == 3:
    print("CUDA is available.")
else:
    print("CUDA is not available.")
```

```
CUDA is available.
```

Use FaceAnalysis to detect faces and filter valid data

In [103…
```
# Initial model
app = FaceAnalysis(allowed_modules=['detection', 'recognition'], det_thresh=0.2, recog_thresh=0.2, model='antelo
app.prepare(ctx_id=0, det_size=(480, 480))

# Check folder
main_folders = ['Wiki_Face_Small_Cleaned', 'Wiki_Face_Small_Noise']
for main_folder in main_folders:
    for i in range(100):
        sub_folder = f"{i:02d}"
        full_path = os.path.join(main_folder, sub_folder)
        make_folder(full_path)

# Detect images and save in the folders
for img_path in tqdm(image_paths, desc="Detecting Faces:"):
    img = load_image(img_path)
    has_face = detect_faces(img, app)
    age, path = calculate_ages(img_path)
    if has_face:
        cv2.imwrite("Wiki_Face_Small_Cleaned/" + path, img)
    else:
        cv2.imwrite("Wiki_Face_Small_Noise/" + path, img)
```

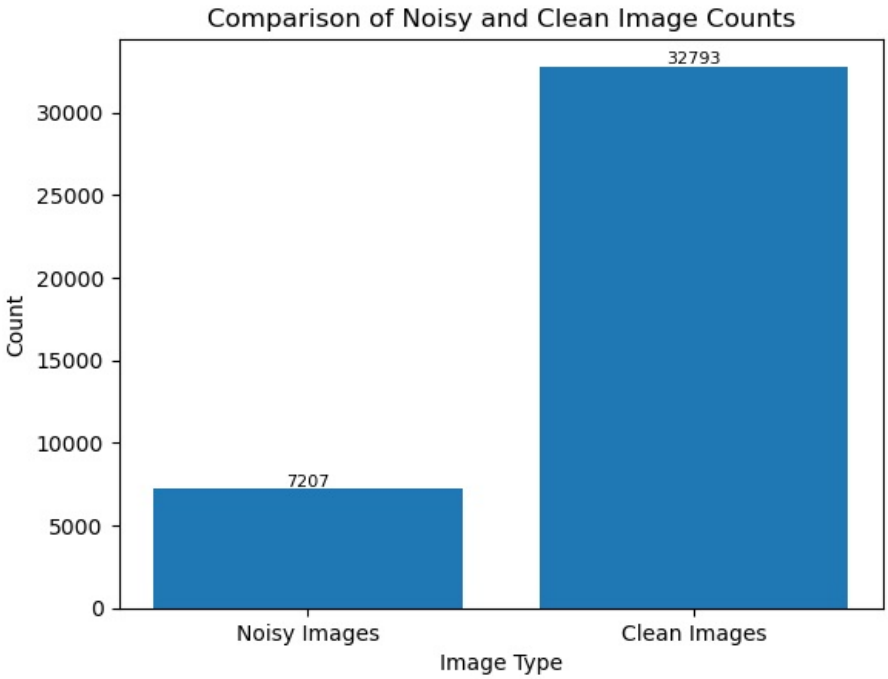
```
Applied providers: ['CUDAExecutionProvider', 'CPUExecutionProvider'], with options: {'CPUExecutionProvider': {},
'CUDAExecutionProvider': {'sdpa_kernel': '0', 'use_tf32': '1', 'prefer_nhwc': '0', 'tunable_op_max_tuning_durati
on_ms': '0', 'enable_skip_layer_norm_strict_mode': '0', 'tunable_op_tuning_enable': '0', 'tunable_op_enable': '0
', 'use_ep_level_unified_stream': '0', 'device_id': '0', 'has_user_compute_stream': '0', 'gpu_external_empty_cac
he': '0', 'cudnn_conv_algo_search': 'EXHAUSTIVE', 'cudnn_conv1d_pad_to_nc1d': '0', 'gpu_mem_limit': '18446744073
709551615', 'gpu_external_alloc': '0', 'gpu_external_free': '0', 'arena_extend_strategy': 'kNextPowerOfTwo', 'do
_copy_in_default_stream': '1', 'enable_cuda_graph': '0', 'user_compute_stream': '0', 'cudnn_conv_use_max_workspa
ce': '1'}}
model ignore: /home/xiaoxin/.insightface/models/buffalo_l/1k3d68.onnx landmark_3d_68
Applied providers: ['CUDAExecutionProvider', 'CPUExecutionProvider'], with options: {'CPUExecutionProvider': {},
'CUDAExecutionProvider': {'sdpa_kernel': '0', 'use_tf32': '1', 'prefer_nhwc': '0', 'tunable_op_max_tuning_durati
on_ms': '0', 'enable_skip_layer_norm_strict_mode': '0', 'tunable_op_tuning_enable': '0', 'tunable_op_enable': '0
', 'use_ep_level_unified_stream': '0', 'device_id': '0', 'has_user_compute_stream': '0', 'gpu_external_empty_cac
he': '0', 'cudnn_conv_algo_search': 'EXHAUSTIVE', 'cudnn_conv1d_pad_to_nc1d': '0', 'gpu_mem_limit': '18446744073
709551615', 'gpu_external_alloc': '0', 'gpu_external_free': '0', 'arena_extend_strategy': 'kNextPowerOfTwo', 'do
_copy_in_default_stream': '1', 'enable_cuda_graph': '0', 'user_compute_stream': '0', 'cudnn_conv_use_max_workspa
ce': '1'}}
model ignore: /home/xiaoxin/.insightface/models/buffalo_l/2d106det.onnx landmark_2d_106
Applied providers: ['CUDAExecutionProvider', 'CPUExecutionProvider'], with options: {'CPUExecutionProvider': {},
'CUDAExecutionProvider': {'sdpa_kernel': '0', 'use_tf32': '1', 'prefer_nhwc': '0', 'tunable_op_max_tuning_durati
on_ms': '0', 'enable_skip_layer_norm_strict_mode': '0', 'tunable_op_tuning_enable': '0', 'tunable_op_enable': '0
', 'use_ep_level_unified_stream': '0', 'device_id': '0', 'has_user_compute_stream': '0', 'gpu_external_empty_cac
he': '0', 'cudnn_conv_algo_search': 'EXHAUSTIVE', 'cudnn_conv1d_pad_to_nc1d': '0', 'gpu_mem_limit': '18446744073
709551615', 'gpu_external_alloc': '0', 'gpu_external_free': '0', 'arena_extend_strategy': 'kNextPowerOfTwo', 'do
_copy_in_default_stream': '1', 'enable_cuda_graph': '0', 'user_compute_stream': '0', 'cudnn_conv_use_max_workspa
ce': '1'}}
find model: /home/xiaoxin/.insightface/models/buffalo_l/det_10g.onnx detection [1, 3, '?', '?'] 127.5 128.0
Applied providers: ['CUDAExecutionProvider', 'CPUExecutionProvider'], with options: {'CPUExecutionProvider': {},
'CUDAExecutionProvider': {'sdpa_kernel': '0', 'use_tf32': '1', 'prefer_nhwc': '0', 'tunable_op_max_tuning_durati
on_ms': '0', 'enable_skip_layer_norm_strict_mode': '0', 'tunable_op_tuning_enable': '0', 'tunable_op_enable': '0
', 'use_ep_level_unified_stream': '0', 'device_id': '0', 'has_user_compute_stream': '0', 'gpu_external_empty_cac
he': '0', 'cudnn_conv_algo_search': 'EXHAUSTIVE', 'cudnn_conv1d_pad_to_nc1d': '0', 'gpu_mem_limit': '18446744073
709551615', 'gpu_external_alloc': '0', 'gpu_external_free': '0', 'arena_extend_strategy': 'kNextPowerOfTwo', 'do
_copy_in_default_stream': '1', 'enable_cuda_graph': '0', 'user_compute_stream': '0', 'cudnn_conv_use_max_workspa
ce': '1'}}
model ignore: /home/xiaoxin/.insightface/models/buffalo_l/genderage.onnx genderage
Applied providers: ['CUDAExecutionProvider', 'CPUExecutionProvider'], with options: {'CPUExecutionProvider': {},
'CUDAExecutionProvider': {'sdpa_kernel': '0', 'use_tf32': '1', 'prefer_nhwc': '0', 'tunable_op_max_tuning_durati
on_ms': '0', 'enable_skip_layer_norm_strict_mode': '0', 'tunable_op_tuning_enable': '0', 'tunable_op_enable': '0
', 'use_ep_level_unified_stream': '0', 'device_id': '0', 'has_user_compute_stream': '0', 'gpu_external_empty_cac
he': '0', 'cudnn_conv_algo_search': 'EXHAUSTIVE', 'cudnn_conv1d_pad_to_nc1d': '0', 'gpu_mem_limit': '18446744073
709551615', 'gpu_external_alloc': '0', 'gpu_external_free': '0', 'arena_extend_strategy': 'kNextPowerOfTwo', 'do
_copy_in_default_stream': '1', 'enable_cuda_graph': '0', 'user_compute_stream': '0', 'cudnn_conv_use_max_workspa
ce': '1'}}
find model: /home/xiaoxin/.insightface/models/buffalo_l/w600k_r50.onnx recognition ['None', 3, 112, 112] 127.5 1
27.5
set det-size: (480, 480)
Detecting Faces:: 100%|██████████| 40000/40000 [05:40<00:00, 117.46it/s]
```

In [7]:
```
# Display top 30 cleaned images
clean_image_paths = get_image_paths("Wiki_Face_Small_Cleaned")
image_show(clean_image_paths)
```

In [8]:
```
# Display top 30 noisy images
noisy_image_paths = get_image_paths("Wiki_Face_Small_Noise")
image_show(noisy_image_paths)
```

In [9]:
```
# Visualization
def display_counts_after_cleaning(clean_image_paths, noisy_image_paths, title):
    noisy_count = len(noisy_image_paths)
    clean_count = len(clean_image_paths)
```

```
    labels = ['Noisy Images', 'Clean Images']
    counts = [noisy_count, clean_count]

    bars = plt.bar(labels, counts)

    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height, str(height), ha='center', va='bottom', fontsize=8)

    plt.title(title)
    plt.xlabel('Image Type')
    plt.ylabel('Count')

    plt.show()

display_counts_after_cleaning(clean_image_paths, noisy_image_paths, 'Comparison of Noisy and Clean Image Counts
```



2.2.2 Use mat file to filter vaild data again

Load mat data and display information

In [10]:
```
mat_data = scipy.io.loadmat('Wiki_Face_Small/wiki_face_small/wiki_face_small.mat')
```

In [11]:
```
print(mat_data.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'wiki'])
```

In [12]:
```
print("Type   of 'wiki' data:", type(mat_data['wiki']))
print("Dtype  of 'wiki' data:", mat_data['wiki'].dtype)
print("Shape  of 'wiki' data:", mat_data['wiki'].shape)
print("Number of dimensions (ndim) of 'wiki' data:", mat_data['wiki'].ndim)
print("Field names of 'wiki' data:", list(mat_data['wiki'].dtype.names))
```

```
Type   of 'wiki' data: <class 'numpy.ndarray'>
Dtype  of 'wiki' data: [('dob', 'O'), ('photo_taken', 'O'), ('full_path', 'O'), ('gender', 'O'), ('name', 'O'),
('face_location', 'O'), ('face_score', 'O'), ('second_face_score', 'O'), ('age', 'O')]
Shape  of 'wiki' data: (1, 1)
Number of dimensions (ndim) of 'wiki' data: 2
Field names of 'wiki' data: ['dob', 'photo_taken', 'full_path', 'gender', 'name', 'face_location', 'face_score',
'second_face_score', 'age']
```

Convert to dataframe

In [13]:
```
#Unpack the data from matlab format data file
data = mat_data['wiki'][0][0]
dob = data[0][0]
photo_taken = data[1][0]
full_path = data[2][0]
gender = data[3][0]
name = data[4][0]
face_location = data[5][0]
face_score = data[6][0]
second_face_score = data[7][0]
```

```
age = data[8][0]
#Build a dictionary for pd
wiki_dict = {
    'dob': dob,
    'photo_taken': photo_taken,
    'full_path': full_path,
    'gender': gender,
    'name': name,
    'face_score': face_score,
    'second_face_score': second_face_score,
    'age': age
}
wiki = pd.DataFrame(wiki_dict)
wiki['full_path'] = wiki['full_path'].apply(lambda x: x[0])
wiki.head()
```

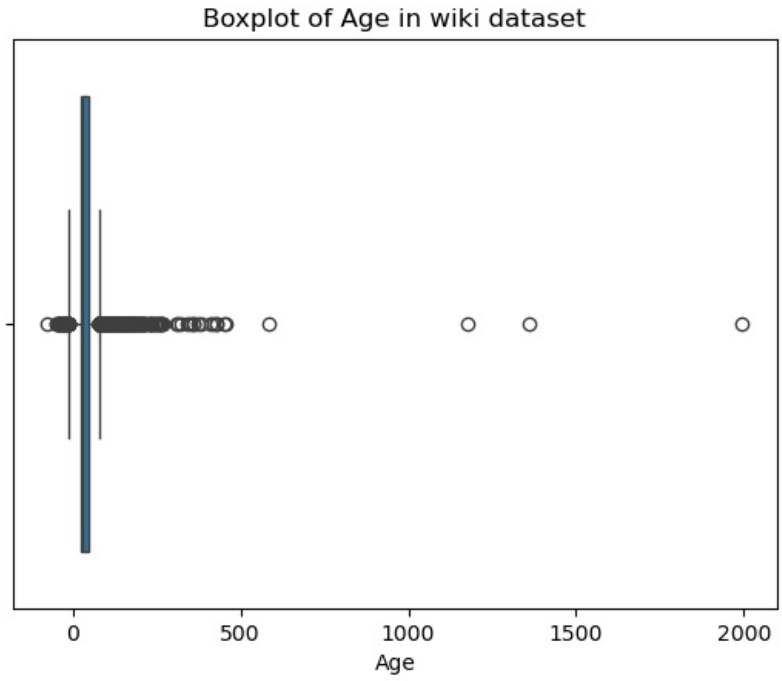
Out[13]:

	dob	photo_taken	full_path	gender	name	face_score	second_face_score	age
0	712934	2012	27/5137227_1951-12-12_2012.jpg	1.0	[Tim McClelland]	2.209695	NaN	60
1	725616	2011	68/22983068_1986-09-01_2011.jpg	1.0	[Antonio Coleman]	-inf	NaN	24
2	722957	2004	09/10801009_1979-05-22_2004.jpg	1.0	[Alexei Beletski]	4.954108	NaN	24
3	706373	1964	15/7775715_1933-12-25_1964.jpg	1.0	[Basil Heatley]	2.480820	1.952827	30
4	725707	2014	19/8679319_1986-12-01_2014.jpg	1.0	[DeSean Jackson]	4.163307	NaN	27

Labels distribution analysis

In [14]:

```
sns.boxplot(x=wiki['age'].dropna())
plt.title('Boxplot of Age in wiki dataset')
plt.xlabel('Age')
plt.show()
```



Data cleaning with special conditions

1. age >= 0 and <= 130
2. second_face_score < 8 or == None

In [15]:

```
clean_image_paths = get_image_paths("Wiki_Face_Small_Cleaned")
def transform_image_paths(path):
    path = path.replace('Wiki_Face_Small_Cleaned\\', '').replace('\\', '/')
    return path

clean_image_paths = pd.Series(clean_image_paths).apply(transform_image_paths)
clean_image_paths = clean_image_paths.tolist()
print('Format of clean_image_paths:', clean_image_paths[0])
```

```
Format of clean_image_paths: 00/10049200_1891-09-16_1958.jpg
```

In [16]:
```
face_cleaned_wiki = wiki[wiki['full_path'].isin(clean_image_paths)]
```

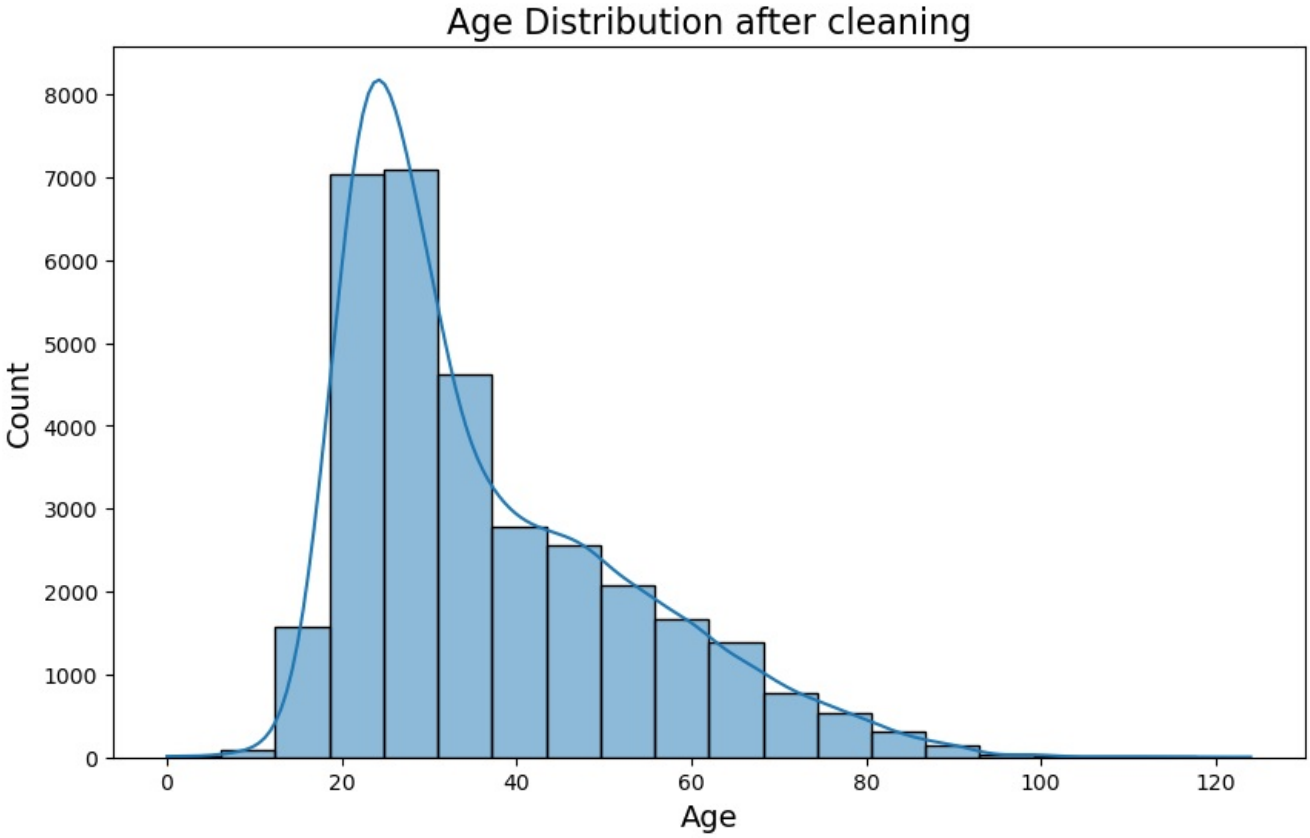
In [17]:
```
double_cleaned_wiki = face_cleaned_wiki[(face_cleaned_wiki['age'] >= 0) & (face_cleaned_wiki['age'] <= 130)
                        & ((face_cleaned_wiki['second_face_score'] < 8) | (face_cleaned_wiki['second_face_score
double_cleaned_wiki = double_cleaned_wiki.reset_index(drop=True)
```

In [18]:
```
double_cleaned_wiki.loc[double_cleaned_wiki['gender'].isna(), 'gender'] = -1
print(len(double_cleaned_wiki[double_cleaned_wiki['gender'] == -1]))
double_cleaned_wiki.loc[double_cleaned_wiki['face_score'] == -np.inf, 'face_score'] = -100
print(len(double_cleaned_wiki[double_cleaned_wiki['face_score'] == -100]))
double_cleaned_wiki.loc[double_cleaned_wiki['second_face_score'].isna(), 'second_face_score'] = -100
print(len(double_cleaned_wiki[double_cleaned_wiki['second_face_score'] == -100]))
```

```
659
4413
30112
```

Display age distribution after cleaning

In [19]:
```
plt.figure(figsize=(10, 6))
sns.histplot(double_cleaned_wiki['age'].dropna(), bins=20, kde=True)

plt.title('Age Distribution after cleaning', fontsize=16)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Count', fontsize=14)

plt.show()
```

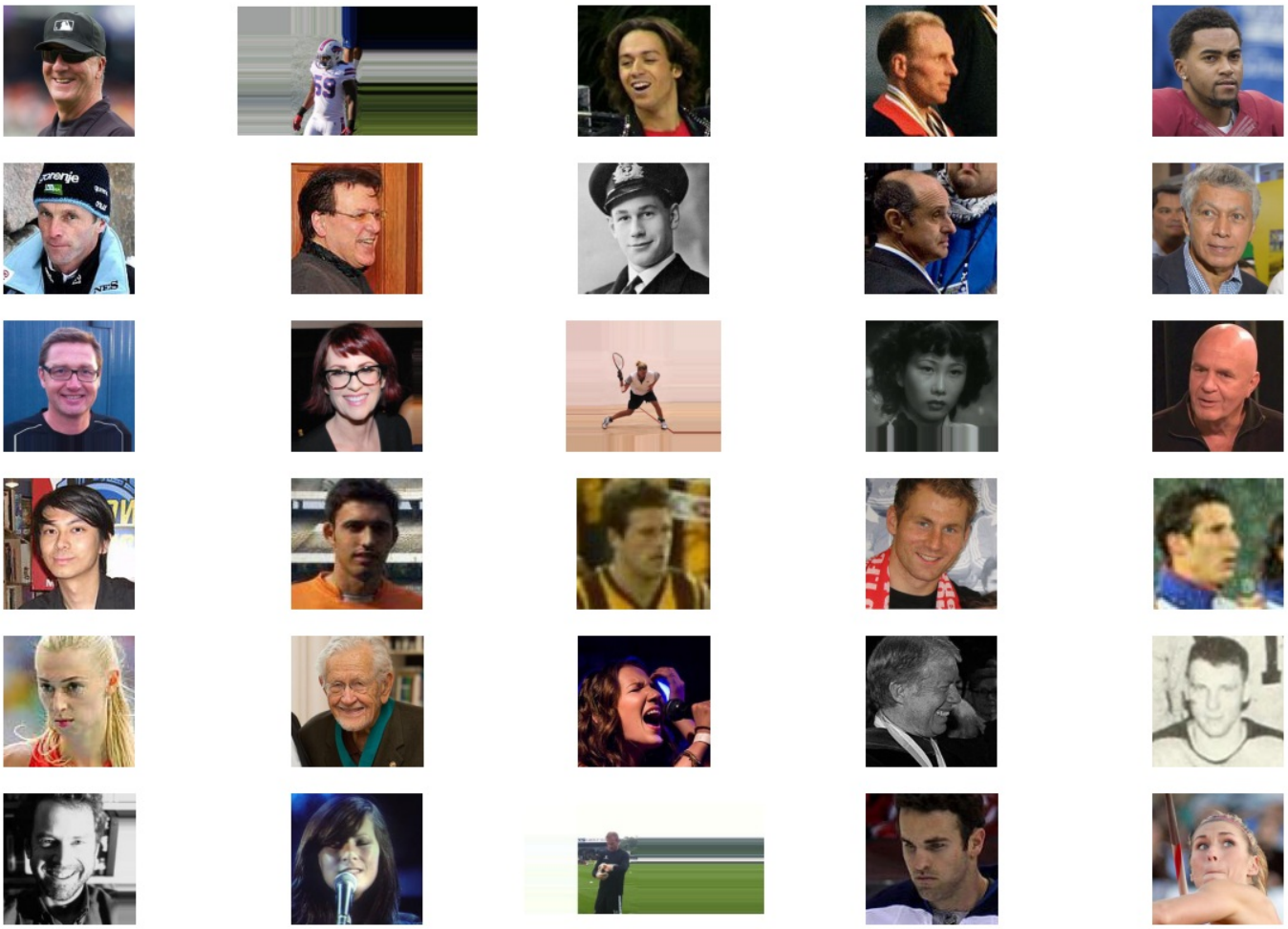


Display clean data and noise data after double cleaning

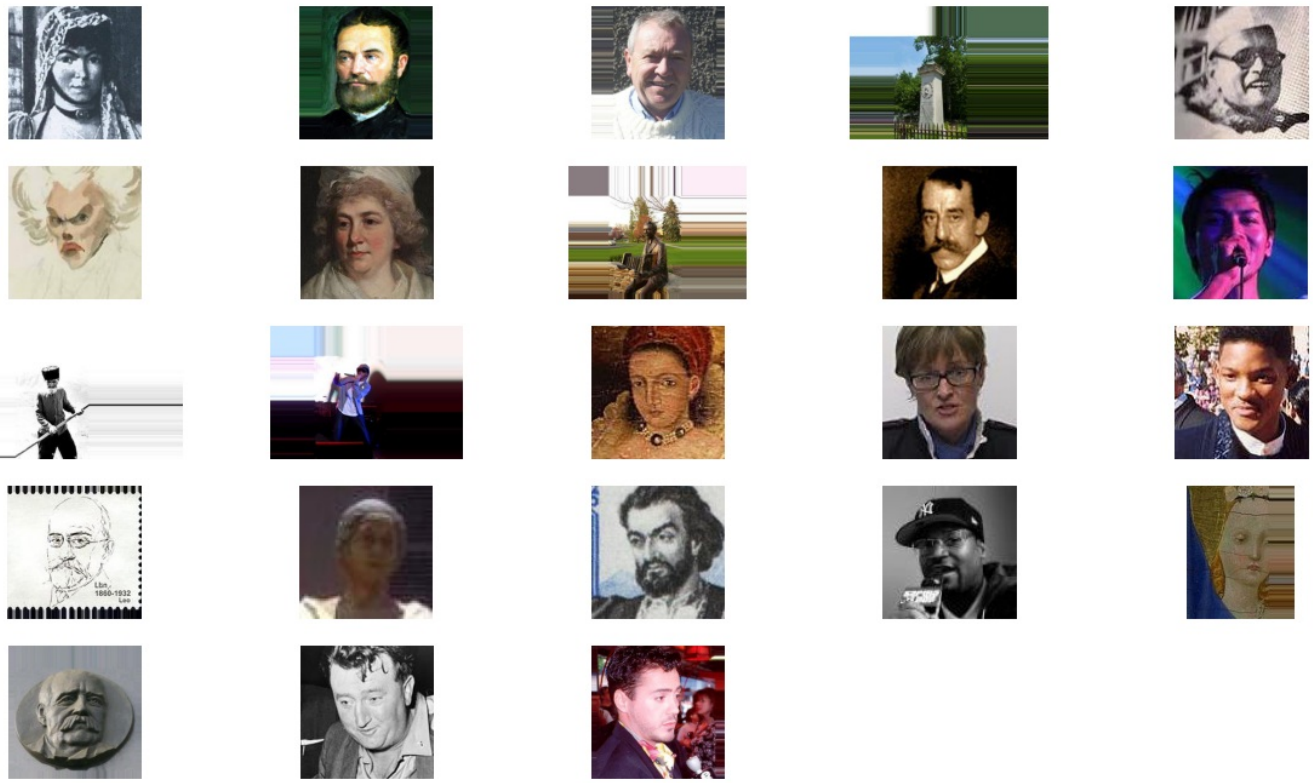
In [20]:
```
double_throwed_wiki = face_cleaned_wiki[~face_cleaned_wiki['full_path'].isin(double_cleaned_wiki['full_path'])]
```

In [21]:
```
double_cleaned_paths = get_image_paths_from_df('Wiki_Face_Small_Cleaned/', double_cleaned_wiki['full_path'])
double_throwed_paths = get_image_paths_from_df('Wiki_Face_Small_Cleaned/', double_throwed_wiki['full_path'])
```

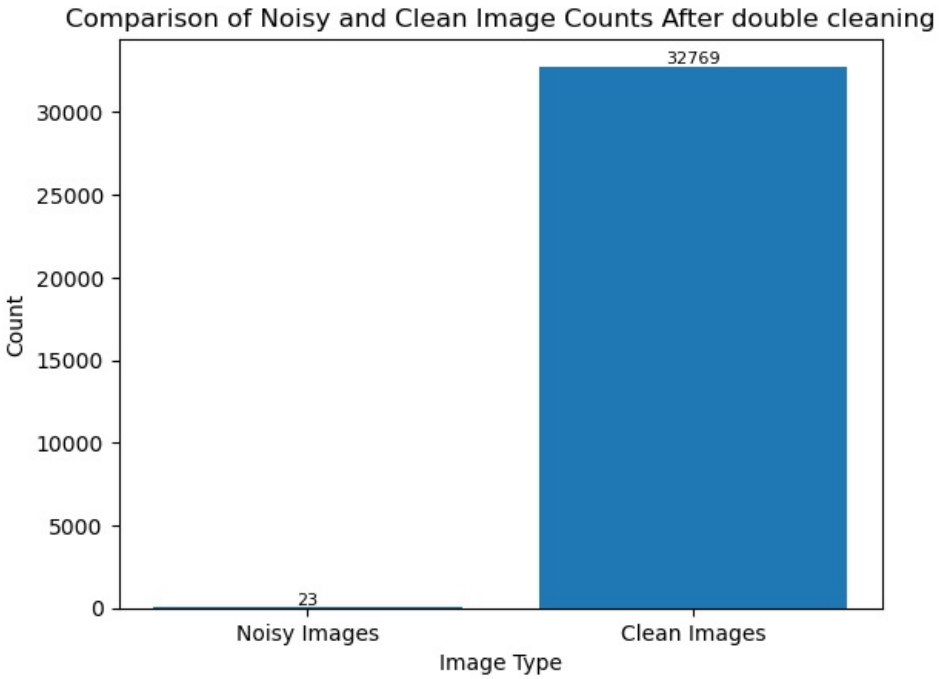
In [22]:
```
image_show(double_cleaned_paths)
```



In [23]:
```
image_show(double_throwed_paths)
```



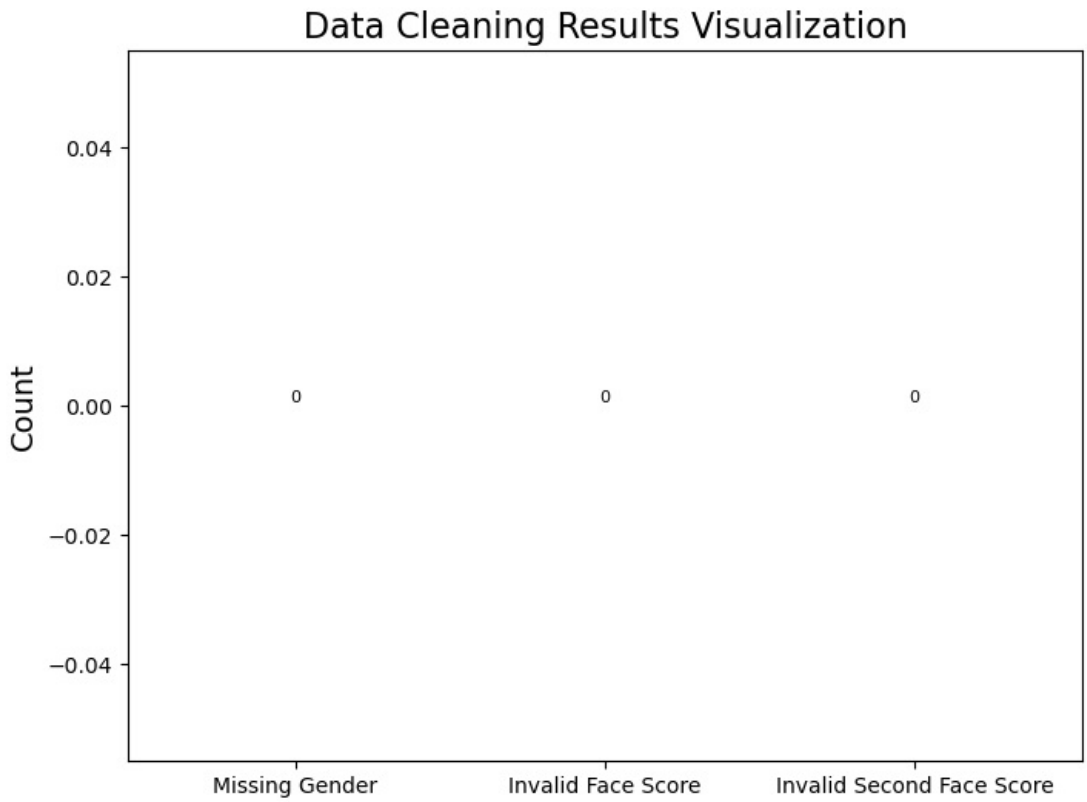
In [24]:
```
display_counts_after_cleaning(double_cleaned_paths, double_throwed_paths,
                              'Comparison of Noisy and Clean Image Counts After double cleaning')
```



2.3 Data exploration again after data cleaning

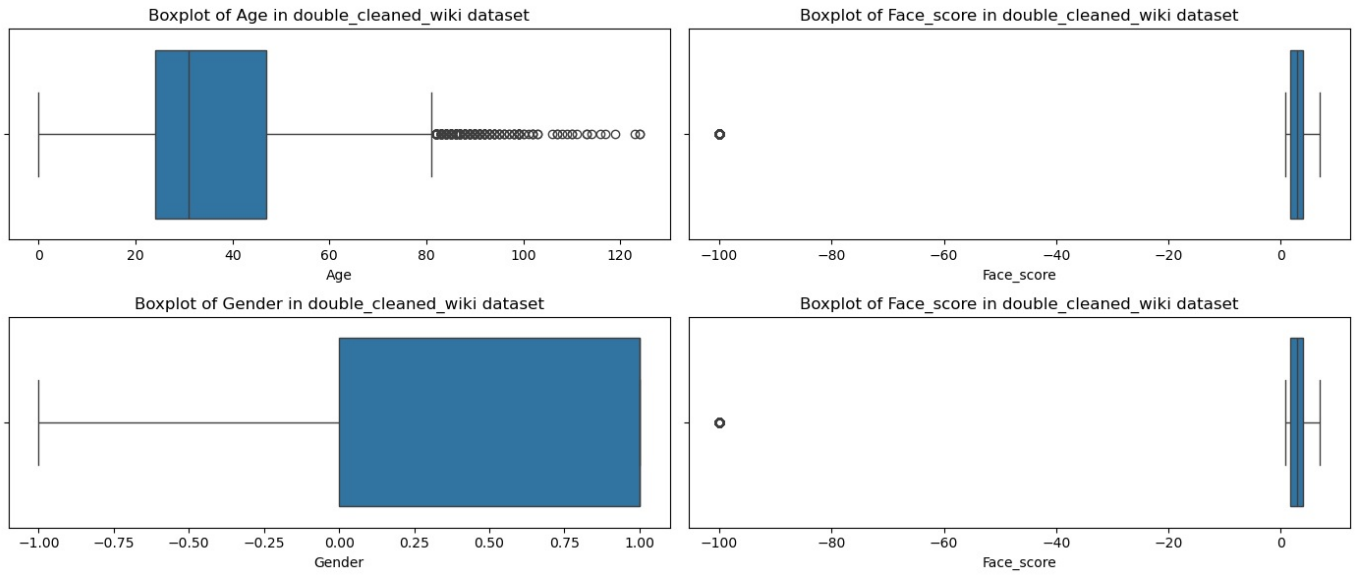
Invalid value exploration

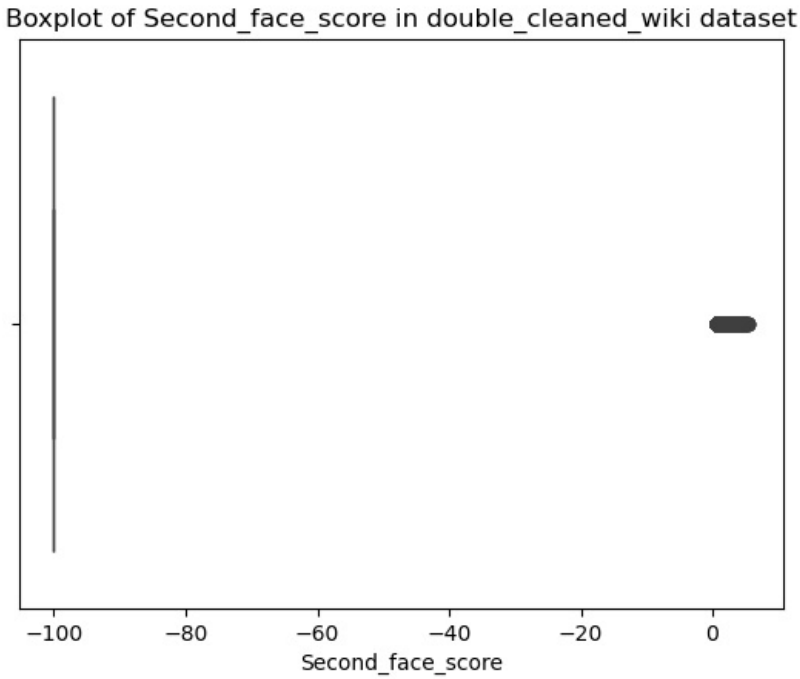
In [25]:
```
def visualize_invalid_value(df):
    gender_missing_count = len(df.loc[double_cleaned_wiki['gender'].isna(), 'gender'])
    face_score_invalid_count = len(df.loc[double_cleaned_wiki['face_score'] == -np.inf, 'face_score'])
    second_face_score_invalid_count = len(df.loc[double_cleaned_wiki['second_face_score'].isna(), 'second_face_

    labels = ['Missing Gender', 'Invalid Face Score', 'Invalid Second Face Score']
    counts = [gender_missing_count, face_score_invalid_count, second_face_score_invalid_count]

    plt.figure(figsize=(8, 6))
    bars = plt.bar(labels, counts)

    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height, str(height), ha='center', va='bottom', fontsize=8)

    plt.title('Data Cleaning Results Visualization', fontsize=16)
    plt.ylabel('Count', fontsize=14)

    plt.show()

visualize_invalid_value(double_cleaned_wiki)
```



Distributions of attributes

In [26]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 6))

sns.boxplot(x=double_cleaned_wiki['age'].dropna(), ax=axes[0][0])
axes[0][0].set_title('Boxplot of Age in double_cleaned_wiki dataset')
axes[0][0].set_xlabel('Age')

sns.boxplot(x=double_cleaned_wiki['face_score'].replace(-np.inf, np.nan).dropna(), ax=axes[0][1])
axes[0][1].set_title('Boxplot of Face_score in double_cleaned_wiki dataset')
axes[0][1].set_xlabel('Face_score')

sns.boxplot(x=double_cleaned_wiki['gender'].dropna(), ax=axes[1][0])
axes[1][0].set_title('Boxplot of Gender in double_cleaned_wiki dataset')
axes[1][0].set_xlabel('Gender')

sns.boxplot(x=double_cleaned_wiki['face_score'].replace(-np.inf, np.nan).dropna(), ax=axes[1][1])
axes[1][1].set_title('Boxplot of Face_score in double_cleaned_wiki dataset')
axes[1][1].set_xlabel('Face_score')
plt.tight_layout()
plt.show()

sns.boxplot(x=double_cleaned_wiki['second_face_score'])
plt.title('Boxplot of Second_face_score in double_cleaned_wiki dataset')
plt.xlabel('Second_face_score')
plt.show()
```





In [28]:
```
max(double_cleaned_wiki['second_face_score'])
min(double_cleaned_wiki['second_face_score'])
sfs_Q1 = np.percentile(double_cleaned_wiki['second_face_score'].dropna(), 25)
sfs_Q3 = np.percentile(double_cleaned_wiki['second_face_score'].dropna(), 75)
sfs_IQR = sfs_Q3 - sfs_Q1

#outlier boundaries
sfs_lower_bound = sfs_Q1 - 1.5 * sfs_IQR
sfs_upper_bound = sfs_Q3 + 1.5 * sfs_IQR
print(f"Lower bound: {sfs_lower_bound}")
print(f"Upper bound: {sfs_upper_bound}")
```

```
Lower bound: -100.0
Upper bound: -100.0
```

2.4 Data preprocessing: calculate ages, split train dataset and test dataset

Calculate ages

In [27]:
```
output_folder = "Age_Folder"
make_folder(output_folder)
for path in tqdm(double_cleaned_paths, desc="Age Processing"):
    relative_path = path.replace("Wiki_Face_Small_Cleaned/", "")

    matched_row = double_cleaned_wiki[double_cleaned_wiki['full_path'] == relative_path]

    if not matched_row.empty:
        age = matched_row['age'].values[0]

        age_folder = os.path.join(output_folder, str(age))
        make_folder(age_folder)

        source_file = path
        destination_file = os.path.join(age_folder, os.path.basename(path))
        shutil.copy(source_file, destination_file)
```

```
Age Processing: 100%|██████████| 32769/32769 [01:15<00:00, 432.58it/s]
```

Remove categories that are too small

Remove categories that are too small to divide the appropriate test set and that can not fully learn age-appropriate features.

In [29]:
```
def delete_folders_with_few_images(main_folders, min_image_count=10):
    image_extensions = ('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff')

    for main_folder in main_folders:
        print(f"Processing main folder: {main_folder}")
```

```
        for root, dirs, files in os.walk(main_folder):
            image_count = sum(1 for file in files if file.lower().endswith(image_extensions))

            if image_count < min_image_count and root != main_folder:
                print(f"Deleting folder: {root}, it contains {image_count} images")
                shutil.rmtree(root)
```

In [29]:
```
delete_folders_with_few_images([output_folder], min_image_count=10)
```

```
Processing main folder: Age_Folder
Deleting folder: Age_Folder\0, it contains 6 images
Deleting folder: Age_Folder\1, it contains 1 images
Deleting folder: Age_Folder\100, it contains 2 images
Deleting folder: Age_Folder\101, it contains 1 images
Deleting folder: Age_Folder\102, it contains 3 images
Deleting folder: Age_Folder\103, it contains 2 images
Deleting folder: Age_Folder\106, it contains 1 images
Deleting folder: Age_Folder\107, it contains 2 images
Deleting folder: Age_Folder\108, it contains 1 images
Deleting folder: Age_Folder\109, it contains 1 images
Deleting folder: Age_Folder\110, it contains 2 images
Deleting folder: Age_Folder\111, it contains 1 images
Deleting folder: Age_Folder\113, it contains 2 images
Deleting folder: Age_Folder\114, it contains 1 images
Deleting folder: Age_Folder\116, it contains 1 images
Deleting folder: Age_Folder\117, it contains 1 images
Deleting folder: Age_Folder\119, it contains 1 images
Deleting folder: Age_Folder\123, it contains 1 images
Deleting folder: Age_Folder\124, it contains 2 images
Deleting folder: Age_Folder\2, it contains 1 images
Deleting folder: Age_Folder\3, it contains 3 images
Deleting folder: Age_Folder\4, it contains 1 images
Deleting folder: Age_Folder\5, it contains 3 images
Deleting folder: Age_Folder\6, it contains 4 images
Deleting folder: Age_Folder\7, it contains 7 images
Deleting folder: Age_Folder\9, it contains 9 images
Deleting folder: Age_Folder\93, it contains 7 images
Deleting folder: Age_Folder\94, it contains 6 images
Deleting folder: Age_Folder\95, it contains 4 images
Deleting folder: Age_Folder\96, it contains 3 images
Deleting folder: Age_Folder\97, it contains 2 images
Deleting folder: Age_Folder\98, it contains 4 images
```

Load dataset and split test dataset and train dataset

In [30]:
```
random_seed = 42
random.seed(random_seed)

image_dir = "./Age_Folder"
train_dir = "./train_set"
test_dir = "./test_set"
val_dir = './val_set'
```

In [31]:
```
make_folder(train_dir)
make_folder(test_dir)

# Create train_set, test_set and validation set
for age_folder in os.listdir(image_dir):
    age_path = os.path.join(image_dir, age_folder)

    # Make sure every age folder use the same proportion to split
    if os.path.isdir(age_path):
        images = [img for img in os.listdir(age_path) if img.endswith(('.jpg', '.png', '.jpeg'))]
        random.shuffle(images)
        num_samples = len(images)
        num_train_samples = floor(0.8 * num_samples)
        num_val_samples = floor(0.1 * num_samples)
        num_test_samples = num_samples - num_train_samples - num_val_samples

        train_imgs = images[:num_train_samples]
        val_imgs = images[num_train_samples:num_train_samples + num_val_samples]
        test_imgs = images[num_train_samples + num_val_samples:]

        make_folder(os.path.join(train_dir, age_folder))
        make_folder(os.path.join(test_dir, age_folder))
        make_folder(os.path.join(val_dir, age_folder))

        for img in train_imgs:
            shutil.copy(os.path.join(age_path, img), os.path.join(train_dir, age_folder, img))

        for img in val_imgs:
```

```
            shutil.copy(os.path.join(age_path, img), os.path.join(val_dir, age_folder, img))

        for img in test_imgs:
            shutil.copy(os.path.join(age_path, img), os.path.join(test_dir, age_folder, img))
```

In [31]:
```
double_cleaned_wiki['full_path'] = double_cleaned_wiki['full_path'].apply(lambda x: 'Wiki_Face_Small_Cleaned\\'
double_cleaned_wiki
```

Out[31]:

	dob	photo_taken	full_path	gender	name	face_score	second_fac
0	712934	2012	Wiki_Face_Small_Cleaned\27/5137227_1951-12-12_...	1.0	[Tim McClelland]	2.209695	-100
1	725616	2011	Wiki_Face_Small_Cleaned\68/22983068_1986-09-01...	1.0	[Antonio Coleman]	-100.000000	-100
2	722957	2004	Wiki_Face_Small_Cleaned\09/10801009_1979-05-22...	1.0	[Alexei Beletski]	4.954108	-100
3	706373	1964	Wiki_Face_Small_Cleaned\15/7775715_1933-12-25_...	1.0	[Basil Heatley]	2.480820	1
4	725707	2014	Wiki_Face_Small_Cleaned\19/8679319_1986-12-01_...	1.0	[DeSean Jackson]	4.163307	-100
...	...	...	...	...	...	...	
32764	725690	2015	Wiki_Face_Small_Cleaned\74/42456374_1986-11-14...	1.0	[Cory Michael Smith]	3.986726	-100
32765	715408	2011	Wiki_Face_Small_Cleaned\65/3354265_1958-09-20_...	1.0	[Norbert Meier]	2.972500	-100
32766	723292	2013	Wiki_Face_Small_Cleaned\12/42209512_1980-04-21...	0.0	[Rebecca Zlotowski]	3.559609	-100
32767	727788	2011	Wiki_Face_Small_Cleaned\25/34014025_1992-08-12...	1.0	[Conor McAleny]	3.430896	-100
32768	726413	2014	Wiki_Face_Small_Cleaned\61/36534961_1988-11-06...	0.0	[Conchita Wurst]	3.024268	-100

32769 rows × 8 columns

In [32]:
```
zeni_splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=540367825)

#First split
zeni_train_val_df, zeni_test_df = train_test_split(double_cleaned_wiki, test_size = 0.2, random_state = 5403678

#split train+val into train and validation
zeni_train_df, zeni_val_df = train_test_split(zeni_train_val_df, test_size = 0.25, random_state = 540367825, st

# create a Dataset from dataframe
from Image_Dataset_From_Df import Image_Dataset_From_Df
#class Image_Dataset_From_Df(Dataset):
#    def __init__(self, dataframe, image_column, label_column, transform = None):
#        self.dataframe = dataframe
#        self.image_column = image_column
#        self.label_column = label_column
#        self.transform = transform
#
#    def __len__(self):
#        return len(self.dataframe)
#
#    def __getitem__(self, idx):
#        img_path = self.dataframe.iloc[idx][self.image_column]
#        label = self.dataframe.iloc[idx][self.label_column]
#        #Open the image and apply transformations
#        image = Image.open(img_path).convert('RGB')
#        if self.transform:
#            image = self.transform(image)
#        return image, label
```

Create DataSet

In [34]:
```
#initialize the transform, the dataset and train_loader
zeni_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
```

```
zeni_dataset = Image_Dataset_From_Df(dataframe = double_cleaned_wiki, image_column = 'full_path', label_column =
zeni_train_loader = DataLoader(zeni_dataset, batch_size = 64, shuffle = True, num_workers = 4)

#Initialize variables to compute mean and std
zeni_mean = torch.zeros(3)
zeni_std = torch.zeros(3)
zeni_n_samples = 0

#compute the sum of means and std
for images, labels in zeni_train_loader:
    batch_samples = images.size(0)
    images = images.view(batch_samples, images.size(1), -1)
    zeni_mean += images.mean(2).sum(0)
    zeni_std += images.std(2).sum(0)
    zeni_n_samples += batch_samples

#Final mean and std
zeni_mean /= zeni_n_samples
zeni_std /= zeni_n_samples

print(f'Mean: {zeni_mean}')
print(f'Standard Deviation: {zeni_std}')
```

```
Mean: tensor([0.4732, 0.4151, 0.3826])
Standard Deviation: tensor([0.2507, 0.2310, 0.2261])
```

In [33]:
```
#preprocessing
zeni_preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.4732, 0.4151, 0.3826], std = [0.2507, 0.2310, 0.2261])
])

train_dataset = Image_Dataset_From_Df(zeni_train_df, image_column = 'full_path', label_column = 'age', transfor
val_dataset = Image_Dataset_From_Df(zeni_val_df, image_column = 'full_path', label_column = 'age', transform =
test_dataset = Image_Dataset_From_Df(zeni_test_df, image_column = 'full_path', label_column = 'age', transform =
```

Calculate mean and std in order to normalize data

In [34]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

3. Model Implementation

Define save function

In [35]:
```
def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
    torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'best params:': best_params,
            'best loss': best_loss,
            'training time': training_time,
            'last val loss': last_val_loss,
        }, checkpoint_name + '.pth')
```

Define training function

In [36]:
```
def training_model(model, device, optimizer_name, lr, epochs, train_dataset, val_dataset, test_dataset, batch_s
    model.to(device)
    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_work
    val_dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers
    test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_worke

    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr) # default

    if optimizer_name == 'Adam':
        optimizer = optim.Adam(model.parameters(), lr=lr)
    elif optimizer_name == 'SGD':
        optimizer = optim.SGD(model.parameters(), lr=lr)
    elif optimizer_name == 'AdamW':
        optimizer = optim.AdamW(model.parameters(), lr=lr)

    start = time.time()
    for epoch in range(epochs):
        model.train()
```

```
        train_loss = 0
        all_train_outputs = []  # To store all outputs
        all_train_labels = []   # To store all labels

        for images, labels in train_dataloader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            #zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass: compute predicted ages by passing image to the model
            outputs = model(images)

            # Calculate the loss(MSE)
            loss = criterion(outputs, labels)
            train_loss += loss.item()

            loss.backward()
            optimizer.step()

            all_train_outputs.append(outputs.cpu().detach().numpy())
            all_train_labels.append(labels.cpu().detach().numpy())

        average_batch_train_loss = train_loss / len(train_dataloader)
        all_train_outputs = np.concatenate(all_train_outputs)
        all_train_labels = np.concatenate(all_train_labels)

        train_mse = np.mean((all_train_outputs - all_train_labels) ** 2)  # Calculate MSE for the entire traini

        model.eval()
        val_loss = 0
        all_val_outputs = []  # To store all outputs
        all_val_labels = []   # To store all labels

        with torch.no_grad():
            for images, labels in val_dataloader:
                images = images.to(device)
                labels = labels.to(device).float().view(-1, 1)

                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                all_val_outputs.append(outputs.cpu().detach().numpy())
                all_val_labels.append(labels.cpu().detach().numpy())

        average_batch_val_loss = val_loss / len(val_dataloader)
        all_val_outputs = np.concatenate(all_val_outputs)
        all_val_labels = np.concatenate(all_val_labels)
        val_mse = np.mean((all_val_outputs - all_val_labels) ** 2)  # Calculate MSE for the entire val set
        time_str = time.asctime(time.localtime(time.time()))
        print(f'{time_str}, Epoch [{epoch + 1}/{epochs}], Train MSE: {train_mse:.4f}, Average batch train mse:
        start = time.time()


    model.eval()
    test_loss = 0
    all_test_outputs = []  # To store all outputs
    all_test_labels = []   # To store all labels
    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            all_test_outputs.append(outputs.cpu().detach().numpy())
            all_test_labels.append(labels.cpu().detach().numpy())
        average_batch_test_loss = test_loss / len(test_dataloader)
        all_test_outputs = np.concatenate(all_test_outputs)
        all_test_labels = np.concatenate(all_test_labels)
        test_mse = np.mean((all_test_outputs - all_test_labels) ** 2)  # Calculate MSE for the entire test set
        time_str = time.asctime(time.localtime(time.time()))
        print(f'{time_str}, Epoch [{epoch + 1}/{epochs}], Train MSE: {train_mse:.4f}, Average batch train mse:
    params = {'epochs': epoch, 'learning_rate': lr, 'batch_size': batch_size,
                                        'optimizer': optimizer_name}
    training_time = time.time() - start
    save(params['epochs'], best_model, optimizer, params, test_mse, training_time, test_mse, checkpoint_name)
```

3.1 Model 1: EfficientNet

In [37]:
```
class EfficientNet(nn.Module):
    def __init__(self):
        super(EfficientNet, self).__init__()
        self.efficientnet = models.efficientnet_b0(weights=None)

        self.efficientnet.classifier[1] = nn.Linear(
            in_features=self.efficientnet.classifier[1].in_features,out_features=1)

    def forward(self, x):
        return self.efficientnet(x)
```

In [40]:
```
efficientModel = EfficientNet()
training_model(efficientModel, device, 'Adam', 0.001, 20, train_dataset, val_dataset, test_dataset, 128)
```

```
Mon Oct 14 04:06:43 2024, Epoch [1/20], Train MSE: 604.3596, Average batch train mse: 603.6266, Val MSE: 260.538
3, Average batch Val loss: 263.4427
Mon Oct 14 04:07:29 2024, Epoch [2/20], Train MSE: 237.5758, Average batch train mse: 237.5671, Val MSE: 212.852
4, Average batch Val loss: 214.2836
Mon Oct 14 04:08:14 2024, Epoch [3/20], Train MSE: 202.1823, Average batch train mse: 202.3077, Val MSE: 188.175
0, Average batch Val loss: 190.2265
Mon Oct 14 04:09:01 2024, Epoch [4/20], Train MSE: 177.0219, Average batch train mse: 176.9184, Val MSE: 176.463
7, Average batch Val loss: 178.5422
Mon Oct 14 04:09:47 2024, Epoch [5/20], Train MSE: 159.4559, Average batch train mse: 159.4488, Val MSE: 160.779
3, Average batch Val loss: 161.6823
Mon Oct 14 04:10:33 2024, Epoch [6/20], Train MSE: 147.3686, Average batch train mse: 147.3275, Val MSE: 160.234
4, Average batch Val loss: 160.4678
Mon Oct 14 04:11:20 2024, Epoch [7/20], Train MSE: 136.4146, Average batch train mse: 136.3061, Val MSE: 163.848
6, Average batch Val loss: 164.6980
Mon Oct 14 04:12:06 2024, Epoch [8/20], Train MSE: 125.2461, Average batch train mse: 125.2325, Val MSE: 150.274
9, Average batch Val loss: 150.6426
Mon Oct 14 04:12:52 2024, Epoch [9/20], Train MSE: 115.4624, Average batch train mse: 115.5974, Val MSE: 147.774
9, Average batch Val loss: 148.3386
Mon Oct 14 04:13:38 2024, Epoch [10/20], Train MSE: 104.3967, Average batch train mse: 104.3547, Val MSE: 141.06
32, Average batch Val loss: 141.4720
Mon Oct 14 04:14:23 2024, Epoch [11/20], Train MSE: 95.2328, Average batch train mse: 95.2260, Val MSE: 148.0763
, Average batch Val loss: 148.3987
Mon Oct 14 04:15:09 2024, Epoch [12/20], Train MSE: 86.8549, Average batch train mse: 86.8710, Val MSE: 155.6426
, Average batch Val loss: 156.4980
Mon Oct 14 04:15:56 2024, Epoch [13/20], Train MSE: 77.1641, Average batch train mse: 77.1726, Val MSE: 152.0254
, Average batch Val loss: 151.8340
Mon Oct 14 04:16:41 2024, Epoch [14/20], Train MSE: 70.4881, Average batch train mse: 70.4424, Val MSE: 153.8999
, Average batch Val loss: 154.1451
Mon Oct 14 04:17:27 2024, Epoch [15/20], Train MSE: 61.9348, Average batch train mse: 61.9231, Val MSE: 159.0884
, Average batch Val loss: 158.7527
Mon Oct 14 04:18:13 2024, Epoch [16/20], Train MSE: 55.4089, Average batch train mse: 55.4126, Val MSE: 152.6528
, Average batch Val loss: 152.4735
Mon Oct 14 04:18:59 2024, Epoch [17/20], Train MSE: 48.4418, Average batch train mse: 48.4459, Val MSE: 157.9023
, Average batch Val loss: 158.3456
Mon Oct 14 04:19:45 2024, Epoch [18/20], Train MSE: 45.3338, Average batch train mse: 45.3342, Val MSE: 151.7619
, Average batch Val loss: 152.3888
Mon Oct 14 04:20:31 2024, Epoch [19/20], Train MSE: 42.1728, Average batch train mse: 42.1664, Val MSE: 150.5354
, Average batch Val loss: 151.1352
Mon Oct 14 04:21:17 2024, Epoch [20/20], Train MSE: 37.6186, Average batch train mse: 37.6584, Val MSE: 151.1316
, Average batch Val loss: 151.8175
Mon Oct 14 04:21:31 2024, Epoch [20/20], Train MSE: 37.6186, Average batch train mse: 37.6584, Test MSE: 146.445
7, Average batch Test loss: 147.9545
```

3.2 Model 2: ResNet18

In [38]:
```
class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        self.resnet = models.resnet18(weights=None)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 1)

    def forward(self, x):
        return self.resnet(x)
```

In [48]:
```
resnetModel = ResNet18()
training_model(resnetModel, device, 'Adam', 0.001, 20, train_dataset, val_dataset, test_dataset, 512)
```

```
Mon Oct 14 04:25:11 2024, Epoch [1/20], Train MSE: 751.1332, Average batch train mse: 743.8762, Val MSE: 318.318
2, Average batch Val loss: 318.0082
Mon Oct 14 04:25:52 2024, Epoch [2/20], Train MSE: 229.5426, Average batch train mse: 229.1841, Val MSE: 218.123
5, Average batch Val loss: 217.9754
Mon Oct 14 04:26:32 2024, Epoch [3/20], Train MSE: 190.3164, Average batch train mse: 189.9536, Val MSE: 225.956
7, Average batch Val loss: 225.8157
Mon Oct 14 04:27:12 2024, Epoch [4/20], Train MSE: 175.5494, Average batch train mse: 175.2832, Val MSE: 175.947
5, Average batch Val loss: 175.8635
Mon Oct 14 04:27:53 2024, Epoch [5/20], Train MSE: 164.2854, Average batch train mse: 163.6801, Val MSE: 451.048
0, Average batch Val loss: 450.8979
Mon Oct 14 04:28:33 2024, Epoch [6/20], Train MSE: 156.4536, Average batch train mse: 156.2723, Val MSE: 160.195
8, Average batch Val loss: 160.0930
Mon Oct 14 04:29:13 2024, Epoch [7/20], Train MSE: 147.9689, Average batch train mse: 147.9194, Val MSE: 160.740
1, Average batch Val loss: 160.6181
Mon Oct 14 04:29:53 2024, Epoch [8/20], Train MSE: 142.2443, Average batch train mse: 142.2850, Val MSE: 161.648
0, Average batch Val loss: 161.5960
Mon Oct 14 04:30:33 2024, Epoch [9/20], Train MSE: 132.5648, Average batch train mse: 132.4020, Val MSE: 159.661
6, Average batch Val loss: 159.7131
Mon Oct 14 04:31:13 2024, Epoch [10/20], Train MSE: 123.7871, Average batch train mse: 123.6819, Val MSE: 166.99
31, Average batch Val loss: 167.0435
Mon Oct 14 04:31:53 2024, Epoch [11/20], Train MSE: 114.3079, Average batch train mse: 113.9074, Val MSE: 157.83
65, Average batch Val loss: 157.9448
Mon Oct 14 04:32:33 2024, Epoch [12/20], Train MSE: 100.9757, Average batch train mse: 101.3094, Val MSE: 157.93
68, Average batch Val loss: 158.0253
Mon Oct 14 04:33:12 2024, Epoch [13/20], Train MSE: 90.0751, Average batch train mse: 90.0017, Val MSE: 181.8019
, Average batch Val loss: 181.8438
Mon Oct 14 04:33:52 2024, Epoch [14/20], Train MSE: 78.2323, Average batch train mse: 78.5692, Val MSE: 161.0913
, Average batch Val loss: 161.2156
Mon Oct 14 04:34:31 2024, Epoch [15/20], Train MSE: 64.5428, Average batch train mse: 64.5099, Val MSE: 163.0558
, Average batch Val loss: 163.4163
Mon Oct 14 04:35:10 2024, Epoch [16/20], Train MSE: 52.4345, Average batch train mse: 52.2650, Val MSE: 154.8165
, Average batch Val loss: 155.0071
Mon Oct 14 04:35:49 2024, Epoch [17/20], Train MSE: 42.0120, Average batch train mse: 41.9687, Val MSE: 154.7063
, Average batch Val loss: 155.0178
Mon Oct 14 04:36:28 2024, Epoch [18/20], Train MSE: 33.8826, Average batch train mse: 33.8899, Val MSE: 152.0446
, Average batch Val loss: 152.1459
Mon Oct 14 04:37:07 2024, Epoch [19/20], Train MSE: 26.3525, Average batch train mse: 26.4130, Val MSE: 154.6877
, Average batch Val loss: 154.8873
Mon Oct 14 04:37:46 2024, Epoch [20/20], Train MSE: 21.5238, Average batch train mse: 21.4718, Val MSE: 158.4879
, Average batch Val loss: 158.8359
Mon Oct 14 04:38:00 2024, Epoch [20/20], Train MSE: 21.5238, Average batch train mse: 21.4718, Test MSE: 162.129
2, Average batch Test loss: 162.3819
```

3.3 Model 3: SwinTransformerWithMLP

In [39]:

```python
class SwinTransformerWithMLP(nn.Module):
    def __init__(self, num_classes=1, hidden_size=512, mlp_layers=3):
        super(SwinTransformerWithMLP, self).__init__()

        # Load Swin Transformer
        self.swin = timm.create_model('swinv2_cr_tiny_224', pretrained=False, num_classes=num_classes)

        # Number of input features from the Swin Transformer output
        num_features = self.swin.num_features

        # Create a list to hold the MLP layers
        layers = []

        # First layer from the Swin output to the first hidden layer
        layers.append(nn.Linear(num_classes, hidden_size))
        layers.append(nn.GELU())  # Activation function

        # Add more MLP layers if requested
        for _ in range(mlp_layers - 1):
            layers.append(nn.Linear(hidden_size, hidden_size))
            layers.append(nn.GELU())

        # Final layer to map to the output ages
        layers.append(nn.Linear(hidden_size, num_classes))

        # Combine all the layers into a Sequential module
        self.mlp = nn.Sequential(*layers)

    def forward(self, x):
        # Pass the input through the custom convolutional layers
        #x = self.conv_layers(x)

        # Extract features using the pre-trained Swin Transformer
        x = self.swin(x)

        # Pass through the custom MLP layers
        x = self.mlp(x)
```

```
        return x
```

In [54]:
```
swinModel = SwinTransformerWithMLP()
training_model(swinModel, device, 'Adam', 0.001, 20, train_dataset, val_dataset, test_dataset, 128)
```

```
Mon Oct 14 04:40:23 2024, Epoch [1/20], Train MSE: 352.1195, Average batch train mse: 352.0580, Val MSE: 269.544
5, Average batch Val loss: 272.4969
Mon Oct 14 04:41:38 2024, Epoch [2/20], Train MSE: 268.1312, Average batch train mse: 268.1507, Val MSE: 262.334
4, Average batch Val loss: 265.1083
Mon Oct 14 04:42:53 2024, Epoch [3/20], Train MSE: 258.1024, Average batch train mse: 258.2085, Val MSE: 248.501
0, Average batch Val loss: 251.1842
Mon Oct 14 04:44:08 2024, Epoch [4/20], Train MSE: 240.2324, Average batch train mse: 240.0999, Val MSE: 234.475
0, Average batch Val loss: 237.5668
Mon Oct 14 04:45:25 2024, Epoch [5/20], Train MSE: 231.2657, Average batch train mse: 231.2710, Val MSE: 227.686
3, Average batch Val loss: 230.6684
Mon Oct 14 04:46:41 2024, Epoch [6/20], Train MSE: 222.1694, Average batch train mse: 222.1189, Val MSE: 216.800
6, Average batch Val loss: 219.5435
Mon Oct 14 04:47:57 2024, Epoch [7/20], Train MSE: 209.7446, Average batch train mse: 209.6150, Val MSE: 209.007
1, Average batch Val loss: 211.8469
Mon Oct 14 04:49:12 2024, Epoch [8/20], Train MSE: 200.3422, Average batch train mse: 200.5129, Val MSE: 218.548
6, Average batch Val loss: 218.8931
Mon Oct 14 04:50:28 2024, Epoch [9/20], Train MSE: 191.1390, Average batch train mse: 191.1807, Val MSE: 194.180
0, Average batch Val loss: 196.3269
Mon Oct 14 04:51:43 2024, Epoch [10/20], Train MSE: 188.2820, Average batch train mse: 188.3082, Val MSE: 194.77
93, Average batch Val loss: 196.5551
Mon Oct 14 04:52:58 2024, Epoch [11/20], Train MSE: 176.1989, Average batch train mse: 176.1919, Val MSE: 192.99
06, Average batch Val loss: 195.0335
Mon Oct 14 04:54:14 2024, Epoch [12/20], Train MSE: 174.1908, Average batch train mse: 174.4052, Val MSE: 209.06
48, Average batch Val loss: 210.8857
Mon Oct 14 04:55:29 2024, Epoch [13/20], Train MSE: 168.1393, Average batch train mse: 168.0247, Val MSE: 186.66
55, Average batch Val loss: 187.4656
Mon Oct 14 04:56:44 2024, Epoch [14/20], Train MSE: 163.1557, Average batch train mse: 163.3270, Val MSE: 188.13
00, Average batch Val loss: 190.5829
Mon Oct 14 04:57:59 2024, Epoch [15/20], Train MSE: 157.1900, Average batch train mse: 157.4387, Val MSE: 193.82
78, Average batch Val loss: 193.8437
Mon Oct 14 04:59:15 2024, Epoch [16/20], Train MSE: 152.1947, Average batch train mse: 152.0865, Val MSE: 185.17
67, Average batch Val loss: 187.6579
Mon Oct 14 05:00:32 2024, Epoch [17/20], Train MSE: 140.1495, Average batch train mse: 140.2469, Val MSE: 202.58
26, Average batch Val loss: 204.0865
Mon Oct 14 05:01:47 2024, Epoch [18/20], Train MSE: 137.5427, Average batch train mse: 137.5761, Val MSE: 202.23
95, Average batch Val loss: 203.9182
Mon Oct 14 05:03:03 2024, Epoch [19/20], Train MSE: 131.6308, Average batch train mse: 131.5738, Val MSE: 190.96
37, Average batch Val loss: 193.3274
Mon Oct 14 05:04:19 2024, Epoch [20/20], Train MSE: 122.7155, Average batch train mse: 122.8883, Val MSE: 189.65
86, Average batch Val loss: 193.5330
Mon Oct 14 05:04:36 2024, Epoch [20/20], Train MSE: 122.7155, Average batch train mse: 122.8883, Test MSE: 183.9
973, Average batch Test loss: 183.4713
```

4. Hyperparameter tuning

Define function

In [40]:
```
def hyperparameter_tuning_with_grid_search(model, train_dataset, val_dataset, param_grid, save_name, converge_t

    best_model = None
    min_mse = float('inf')
    file_path = f'{save_name}.pth'

    if os.path.exists(file_path):
        checkpoint = torch.load(file_path)
        min_mse = checkpoint['best loss']
    best_params = {}
    criterion = nn.MSELoss()


    for epochs in param_grid['epochs']:
        for learning_rate in param_grid['learning_rate']:
            for batch_size in param_grid['batch_size']:
                for optimizer_name in param_grid['optimizer']:
                    if save_name == 'efficientModel':
                        model = EfficientNet()
                    elif save_name == 'resnetModel':
                        model = ResNet18()
                    elif save_name == 'swinModel':
                        model = SwinTransformerWithMLP()
                    model.to(device)
                    start_time = time.time()
                    last_val_loss = float('inf')
                    plateau_times = 0
```

```
                    # define dataloader
                    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle
                    val_dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuffle=Fa

                    # define optimizer
                    optimizer = optim.Adam(model.parameters(), lr=learning_rate) # default
                    if optimizer_name == 'Adam':
                        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
                    elif optimizer_name == 'SGD':
                        optimizer = optim.SGD(model.parameters(), lr=learning_rate)
                    elif optimizer_name == 'AdamW':
                        optimizer = optim.AdamW(model.parameters(), lr=learning_rate)



                    for epoch in range(epochs):
                        train_loss = 0
                        all_train_outputs = []  # To store all outputs
                        all_train_labels = []   # To store all labels
                        model.train()

                        for images, labels in train_dataloader:
                            images = images.to(device)
                            labels = labels.to(device).float().view(-1, 1)

                            #zero the parameter gradients
                            optimizer.zero_grad()

                            # Forward pass: compute predicted ages by passing image to the model
                            outputs = model(images)

                            # Calculate the loss(MSE)
                            loss = criterion(outputs, labels)
                            train_loss += loss.item()

                            loss.backward()
                            optimizer.step()

                            all_train_outputs.append(outputs.cpu().detach().numpy())
                            all_train_labels.append(labels.cpu().detach().numpy())

                        average_batch_train_loss = train_loss / len(train_dataloader)
                        all_train_outputs = np.concatenate(all_train_outputs)
                        all_train_labels = np.concatenate(all_train_labels)

                        train_mse = np.mean((all_train_outputs - all_train_labels) ** 2)  # Calculate MSE for t

                        model.eval()
                        val_labels = []
                        val_preds = []

                        with torch.no_grad():
                            for images, labels in val_dataloader:
                                images = images.to(device)
                                labels = labels.to(device).float().view(-1, 1)

                                outputs = model(images)
                                val_labels.extend(labels.cpu().numpy())
                                val_preds.extend(outputs.cpu().numpy())

                        # calculate mse
                        mse = mean_squared_error(val_labels, val_preds)
                        params = {'epochs': epoch, 'learning_rate': learning_rate, 'batch_size': batch_size,
                                  'optimizer': optimizer_name}

                        time_str = time.asctime(time.localtime(time.time()))
                        print(f'{time_str}' + str(params) + f', Epoch [{epoch + 1}/{epochs}], Train MSE: {train_
                        start = time.time()
                        with open(save_name + '_results.txt', 'a') as file:
                            file.write(f'{time_str}' + str(params) + f', Epoch [{epoch + 1}/{epochs}], Train MS
                        if mse < min_mse:

                            min_mse = mse
                            best_model = model
                            best_params = {'epochs': epoch, 'learning_rate': learning_rate, 'batch_size': batch_
                                           'optimizer': optimizer_name}
                            training_time = time.time() - start_time
                            #def save(epoch, model, optimizer, best_loss, training_time, last_val_loss, checkpo
                            save(best_params['epochs'], best_model, optimizer, best_params, min_mse, training_t
                            time_str = time.asctime(time.localtime(time.time()))
                            print(f'{time_str} best_params upgrade:", {best_params}, min_mse: {min_mse}' )
                            with open(save_name + '_results.txt', 'a') as file:
                                file.write(f'{time_str} best param upgrade: ' + str(best_params) + f', mse:{mse
```

```
                        if converge_threshold:
                            if last_val_loss == float('inf') or abs(last_val_loss - mse) > converge_threshold:
                                plateau_times = 0
                                gap = last_val_loss - mse
                                print(f'Step difference of Loss:{gap:.4f}')
                                last_val_loss = mse
                            elif abs(last_val_loss - mse) < converge_threshold:
                                plateau_times += 1
                                gap = last_val_loss - mse
                                print(f'Step difference of Loss:{gap:.4f} Last validation loss:{last_val_loss}'
                                if plateau_times > 3:
                                    print(f'{time_str} {save_name}, {params} Converge: , min_mse: {mse}' )
                                    with open(save_name + '_results.txt', 'a') as file:
                                        file.write(f'{time_str} {save_name}, {params} Converge: , min_mse: {mse
                                    break

    return best_model, min_mse, best_params
```

4.1 EfficientNet hyperparameter tuning

In [39]:
```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001, 0.0001],
    'batch_size': [128],
    'optimizer': ['Adam', 'AdamW', 'SGD']
}
```

In [ ]:
```
efficientModel = EfficientNet()
eff_best_model, eff_min_mse, eff_best_params = hyperparameter_tuning_with_grid_search(efficientModel, train_data

print("Min Mse:", eff_min_mse)
print("Best Parameters:", eff_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_92580\4218000815.py:8: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 09:10:05 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1/
30], Train MSE: 593.8682, Average batch train mse: 593.2995, Val MSE: 248.8236
Step difference of Loss:inf
Mon Oct 14 09:11:04 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2/
30], Train MSE: 211.2373, Average batch train mse: 211.2799, Val MSE: 200.2137
Step difference of Loss:48.6100
Mon Oct 14 09:11:52 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [3/
30], Train MSE: 183.2782, Average batch train mse: 183.3173, Val MSE: 174.3152
Step difference of Loss:25.8985
Mon Oct 14 09:12:40 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [4/
30], Train MSE: 162.2246, Average batch train mse: 162.1487, Val MSE: 160.4344
Step difference of Loss:13.8808
Mon Oct 14 09:13:28 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [5/
30], Train MSE: 149.1827, Average batch train mse: 149.2764, Val MSE: 153.1073
Step difference of Loss:7.3271
Mon Oct 14 09:14:15 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [6/
30], Train MSE: 135.1620, Average batch train mse: 135.2895, Val MSE: 143.5192
Step difference of Loss:9.5881
Mon Oct 14 09:15:05 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [7/
30], Train MSE: 122.6510, Average batch train mse: 122.5569, Val MSE: 154.3707
Step difference of Loss:-10.8515
Mon Oct 14 09:16:17 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [8/
30], Train MSE: 111.4361, Average batch train mse: 111.3317, Val MSE: 147.3076
Step difference of Loss:7.0631
Mon Oct 14 09:17:10 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [9/
30], Train MSE: 99.2762, Average batch train mse: 99.2726, Val MSE: 161.6021
Step difference of Loss:-14.2945
Mon Oct 14 09:18:02 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [10
/30], Train MSE: 88.5573, Average batch train mse: 88.5774, Val MSE: 148.8813
Step difference of Loss:12.7208
Mon Oct 14 09:19:00 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
1/30], Train MSE: 78.3763, Average batch train mse: 78.5464, Val MSE: 162.1622
Step difference of Loss:-13.2808
Mon Oct 14 09:19:56 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
2/30], Train MSE: 70.2061, Average batch train mse: 70.1970, Val MSE: 153.3928
Step difference of Loss:8.7693
Mon Oct 14 09:20:45 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
```

```
3/30], Train MSE: 60.4104, Average batch train mse: 60.4720, Val MSE: 148.3945
Step difference of Loss:4.9983
Mon Oct 14 09:21:39 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
4/30], Train MSE: 54.7151, Average batch train mse: 54.7357, Val MSE: 160.1215
Step difference of Loss:-11.7270
Mon Oct 14 09:22:38 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
5/30], Train MSE: 49.3566, Average batch train mse: 49.4117, Val MSE: 157.0123
Step difference of Loss:3.1092
Mon Oct 14 09:23:29 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
6/30], Train MSE: 44.8852, Average batch train mse: 44.8767, Val MSE: 151.7458
Step difference of Loss:5.2665
Mon Oct 14 09:24:19 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
7/30], Train MSE: 40.1838, Average batch train mse: 40.1908, Val MSE: 155.2753
Step difference of Loss:-3.5295
Mon Oct 14 09:25:17 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
8/30], Train MSE: 38.0287, Average batch train mse: 38.0828, Val MSE: 152.3598
Step difference of Loss:2.9155
Mon Oct 14 09:26:11 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
9/30], Train MSE: 34.5818, Average batch train mse: 34.6055, Val MSE: 151.4476
Step difference of Loss:0.9123
Mon Oct 14 09:27:03 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
0/30], Train MSE: 31.9688, Average batch train mse: 31.9942, Val MSE: 152.5600
Step difference of Loss:-1.1125
Mon Oct 14 09:27:54 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
1/30], Train MSE: 31.6138, Average batch train mse: 31.6262, Val MSE: 151.1699
Step difference of Loss:1.3901
Mon Oct 14 09:28:58 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
2/30], Train MSE: 30.2557, Average batch train mse: 30.3338, Val MSE: 151.4180
Step difference of Loss:-0.2480
Mon Oct 14 09:29:54 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
3/30], Train MSE: 27.7811, Average batch train mse: 27.7844, Val MSE: 150.7059
Step difference of Loss:0.7121
Mon Oct 14 09:30:52 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
4/30], Train MSE: 25.8989, Average batch train mse: 25.9009, Val MSE: 151.5101
Step difference of Loss:-0.8043
Mon Oct 14 09:32:11 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
5/30], Train MSE: 24.5609, Average batch train mse: 24.5678, Val MSE: 163.6201
Step difference of Loss:-12.1100
Mon Oct 14 09:33:08 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
6/30], Train MSE: 25.0618, Average batch train mse: 25.0809, Val MSE: 149.6160
Step difference of Loss:14.0042
Mon Oct 14 09:34:03 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
7/30], Train MSE: 24.1559, Average batch train mse: 24.1639, Val MSE: 153.4259
Step difference of Loss:-3.8099
Mon Oct 14 09:35:07 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
8/30], Train MSE: 24.0571, Average batch train mse: 24.0508, Val MSE: 148.2639
Step difference of Loss:5.1620
Mon Oct 14 09:36:02 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
9/30], Train MSE: 23.4180, Average batch train mse: 23.4388, Val MSE: 149.9534
Step difference of Loss:-1.6895
Mon Oct 14 09:37:05 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [3
0/30], Train MSE: 20.7334, Average batch train mse: 20.7242, Val MSE: 148.6931
Step difference of Loss:1.2604
Mon Oct 14 09:37:59 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [1
/30], Train MSE: 584.5623, Average batch train mse: 583.6859, Val MSE: 248.0681
Step difference of Loss:inf
Mon Oct 14 09:38:54 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [2
/30], Train MSE: 199.3810, Average batch train mse: 199.3394, Val MSE: 254.6775
Step difference of Loss:-6.6095
Mon Oct 14 09:39:48 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [3
/30], Train MSE: 175.3904, Average batch train mse: 175.4305, Val MSE: 168.1368
Step difference of Loss:86.5407
Mon Oct 14 09:40:59 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [4
/30], Train MSE: 157.2612, Average batch train mse: 157.1723, Val MSE: 154.7578
Step difference of Loss:13.3790
Mon Oct 14 09:41:55 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [5
/30], Train MSE: 144.6477, Average batch train mse: 144.4858, Val MSE: 156.8949
Step difference of Loss:-2.1371
Mon Oct 14 09:42:58 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [6
/30], Train MSE: 133.8496, Average batch train mse: 133.7596, Val MSE: 162.3338
Step difference of Loss:-5.4389
Mon Oct 14 09:44:01 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [7
/30], Train MSE: 122.1287, Average batch train mse: 122.0517, Val MSE: 150.9766
Step difference of Loss:11.3572
Mon Oct 14 09:45:10 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [8
/30], Train MSE: 109.6841, Average batch train mse: 109.6359, Val MSE: 164.7665
Step difference of Loss:-13.7899
Mon Oct 14 09:46:18 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [9
/30], Train MSE: 100.9367, Average batch train mse: 100.9564, Val MSE: 162.1183
Step difference of Loss:2.6482
Mon Oct 14 09:47:20 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [1
0/30], Train MSE: 88.9909, Average batch train mse: 89.0152, Val MSE: 167.1539
Step difference of Loss:-5.0355
```

```
Mon Oct 14 09:48:25 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
11/30], Train MSE: 78.9682, Average batch train mse: 78.9682, Val MSE: 155.9803
Step difference of Loss:11.1735
Mon Oct 14 09:49:37 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
12/30], Train MSE: 70.6298, Average batch train mse: 70.6635, Val MSE: 166.5818
Step difference of Loss:-10.6015
Mon Oct 14 09:51:10 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
13/30], Train MSE: 61.2398, Average batch train mse: 61.2401, Val MSE: 168.6276
Step difference of Loss:-2.0458
Mon Oct 14 09:52:08 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
14/30], Train MSE: 55.2434, Average batch train mse: 55.2624, Val MSE: 165.6795
Step difference of Loss:2.9481
Mon Oct 14 09:53:06 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
15/30], Train MSE: 49.2476, Average batch train mse: 49.2741, Val MSE: 163.2185
Step difference of Loss:2.4611
Mon Oct 14 09:54:03 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
16/30], Train MSE: 45.1377, Average batch train mse: 45.1133, Val MSE: 158.8112
Step difference of Loss:4.4073
Mon Oct 14 09:54:57 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
17/30], Train MSE: 41.9016, Average batch train mse: 41.8798, Val MSE: 162.7789
Step difference of Loss:-3.9678
Mon Oct 14 09:55:53 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
18/30], Train MSE: 39.7113, Average batch train mse: 39.7443, Val MSE: 197.7221
Step difference of Loss:-34.9431
Mon Oct 14 09:56:48 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
19/30], Train MSE: 34.7733, Average batch train mse: 34.7678, Val MSE: 157.6677
Step difference of Loss:40.0544
Mon Oct 14 09:57:46 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
20/30], Train MSE: 32.4958, Average batch train mse: 32.4913, Val MSE: 169.6811
Step difference of Loss:-12.0134
Mon Oct 14 09:58:46 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
21/30], Train MSE: 30.6494, Average batch train mse: 30.6587, Val MSE: 160.4278
Step difference of Loss:9.2533
Mon Oct 14 09:59:42 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
22/30], Train MSE: 30.3865, Average batch train mse: 30.3773, Val MSE: 162.0986
Step difference of Loss:-1.6708
Mon Oct 14 10:00:41 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
23/30], Train MSE: 28.3974, Average batch train mse: 28.3903, Val MSE: 163.3531
Step difference of Loss:-1.2545
Mon Oct 14 10:01:40 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
24/30], Train MSE: 27.6256, Average batch train mse: 27.6435, Val MSE: 158.3442
Step difference of Loss:5.0090
Mon Oct 14 10:02:44 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
25/30], Train MSE: 26.3304, Average batch train mse: 26.3364, Val MSE: 156.2699
Step difference of Loss:2.0743
Mon Oct 14 10:04:07 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
26/30], Train MSE: 25.2693, Average batch train mse: 25.2758, Val MSE: 161.6491
Step difference of Loss:-5.3792
Mon Oct 14 10:05:35 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
27/30], Train MSE: 23.2030, Average batch train mse: 23.2067, Val MSE: 159.9005
Step difference of Loss:1.7486
Mon Oct 14 10:06:47 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
28/30], Train MSE: 23.2653, Average batch train mse: 23.2777, Val MSE: 165.4187
Step difference of Loss:-5.5182
Mon Oct 14 10:07:59 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
29/30], Train MSE: 22.8340, Average batch train mse: 22.8470, Val MSE: 153.5920
Step difference of Loss:11.8267
Mon Oct 14 10:09:05 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
30/30], Train MSE: 22.9409, Average batch train mse: 22.9671, Val MSE: 156.9782
Step difference of Loss:-3.3863
Mon Oct 14 10:10:14 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [1/3
0], Train MSE: 341.5028, Average batch train mse: 341.1396, Val MSE: 331.2274
Step difference of Loss:inf
Mon Oct 14 10:11:43 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2/3
0], Train MSE: 228.0452, Average batch train mse: 228.0582, Val MSE: 235.9149
Step difference of Loss:95.3125
Mon Oct 14 10:13:03 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [3/3
0], Train MSE: 207.3052, Average batch train mse: 207.1590, Val MSE: 209.1028
Step difference of Loss:26.8121
Mon Oct 14 10:14:17 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [4/3
0], Train MSE: 186.7661, Average batch train mse: 186.7301, Val MSE: 255.0901
Step difference of Loss:-45.9873
Mon Oct 14 10:15:49 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [5/3
0], Train MSE: 169.3918, Average batch train mse: 169.3507, Val MSE: 171.6750
Step difference of Loss:83.4150
Mon Oct 14 10:17:04 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [6/3
0], Train MSE: 153.5823, Average batch train mse: 153.8991, Val MSE: 173.7944
Step difference of Loss:-2.1194
Mon Oct 14 10:18:01 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [7/3
0], Train MSE: 141.4682, Average batch train mse: 141.4829, Val MSE: 220.0127
Step difference of Loss:-46.2183
Mon Oct 14 10:19:01 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [8/3
0], Train MSE: 131.2778, Average batch train mse: 131.2665, Val MSE: 160.9311
```

```
Step difference of Loss:59.0816
Mon Oct 14 10:20:01 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [9/3
0], Train MSE: 120.4937, Average batch train mse: 120.5298, Val MSE: 150.2179
Step difference of Loss:10.7131
Mon Oct 14 10:21:01 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [10/
30], Train MSE: 110.5883, Average batch train mse: 110.8620, Val MSE: 175.3878
Step difference of Loss:-25.1699
Mon Oct 14 10:22:00 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [11
/30], Train MSE: 104.4857, Average batch train mse: 104.4584, Val MSE: 146.6014
Step difference of Loss:28.7864
Mon Oct 14 10:22:57 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [12
/30], Train MSE: 94.8104, Average batch train mse: 94.7002, Val MSE: 141.2898
Step difference of Loss:5.3116
Mon Oct 14 10:23:55 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [13
/30], Train MSE: 87.2070, Average batch train mse: 87.2319, Val MSE: 143.5402
Step difference of Loss:-2.2504
Mon Oct 14 10:24:52 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [14
/30], Train MSE: 80.3527, Average batch train mse: 80.4521, Val MSE: 142.8330
Step difference of Loss:0.7072
Mon Oct 14 10:25:48 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [15
/30], Train MSE: 71.9571, Average batch train mse: 71.9312, Val MSE: 145.7475
Step difference of Loss:-2.9145
Mon Oct 14 10:26:47 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [16
/30], Train MSE: 66.6877, Average batch train mse: 66.6658, Val MSE: 162.4983
Step difference of Loss:-16.7508
Mon Oct 14 10:28:17 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [17
/30], Train MSE: 60.1186, Average batch train mse: 60.1086, Val MSE: 162.3329
Step difference of Loss:0.1654
Mon Oct 14 10:29:17 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [18
/30], Train MSE: 56.1986, Average batch train mse: 56.2051, Val MSE: 151.5128
Step difference of Loss:10.8201
Mon Oct 14 10:30:16 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [19
/30], Train MSE: 50.3008, Average batch train mse: 50.3349, Val MSE: 155.6665
Step difference of Loss:-4.1537
Mon Oct 14 10:31:12 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [20
/30], Train MSE: 45.7974, Average batch train mse: 45.7938, Val MSE: 169.2877
Step difference of Loss:-13.6212
Mon Oct 14 10:32:13 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [21
/30], Train MSE: 41.8891, Average batch train mse: 41.9441, Val MSE: 150.9468
Step difference of Loss:18.3409
Mon Oct 14 10:33:08 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [22
/30], Train MSE: 38.3213, Average batch train mse: 38.3479, Val MSE: 145.7341
Step difference of Loss:5.2128
Mon Oct 14 10:34:04 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [23
/30], Train MSE: 35.7210, Average batch train mse: 35.7470, Val MSE: 150.7608
Step difference of Loss:-5.0268
Mon Oct 14 10:35:00 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [24
/30], Train MSE: 32.4874, Average batch train mse: 32.4814, Val MSE: 141.4932
Step difference of Loss:9.2676
Mon Oct 14 10:35:56 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [25
/30], Train MSE: 30.2570, Average batch train mse: 30.2639, Val MSE: 145.6577
Step difference of Loss:-4.1644
Mon Oct 14 10:36:53 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [26
/30], Train MSE: 28.4817, Average batch train mse: 28.4676, Val MSE: 136.9358
Step difference of Loss:8.7218
Mon Oct 14 10:37:49 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [27
/30], Train MSE: 27.7961, Average batch train mse: 27.7696, Val MSE: 142.6464
Step difference of Loss:-5.7106
Mon Oct 14 10:38:35 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [28
/30], Train MSE: 25.9748, Average batch train mse: 26.0027, Val MSE: 147.4039
Step difference of Loss:-4.7575
Mon Oct 14 10:39:22 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [29
/30], Train MSE: 23.1756, Average batch train mse: 23.1890, Val MSE: 140.6342
Step difference of Loss:6.7697
Mon Oct 14 10:40:08 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [30
/30], Train MSE: 22.8554, Average batch train mse: 22.8921, Val MSE: 141.6427
Step difference of Loss:-1.0085
Mon Oct 14 10:40:54 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
/30], Train MSE: 1438.9863, Average batch train mse: 1438.2714, Val MSE: 1255.9103
Step difference of Loss:inf
Mon Oct 14 10:41:40 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
/30], Train MSE: 1131.3528, Average batch train mse: 1131.1837, Val MSE: 883.9863
Step difference of Loss:371.9240
Mon Oct 14 10:42:26 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [3
/30], Train MSE: 841.8195, Average batch train mse: 841.0418, Val MSE: 706.0269
Step difference of Loss:177.9594
Mon Oct 14 10:43:11 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [4
/30], Train MSE: 582.7058, Average batch train mse: 582.4438, Val MSE: 423.1359
Step difference of Loss:282.8910
Mon Oct 14 10:43:56 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [5
/30], Train MSE: 383.3571, Average batch train mse: 382.8581, Val MSE: 274.3182
Step difference of Loss:148.8176
Mon Oct 14 10:44:41 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [6
```

```
/30], Train MSE: 247.0093, Average batch train mse: 247.0988, Val MSE: 277.1599
Step difference of Loss:-2.8417
Mon Oct 14 10:45:27 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [7
/30], Train MSE: 169.9951, Average batch train mse: 169.9587, Val MSE: 243.3712
Step difference of Loss:33.7887
Mon Oct 14 10:46:11 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [8
/30], Train MSE: 125.0704, Average batch train mse: 124.9898, Val MSE: 222.9769
Step difference of Loss:20.3944
Mon Oct 14 10:46:56 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [9
/30], Train MSE: 101.6436, Average batch train mse: 101.7676, Val MSE: 214.4179
Step difference of Loss:8.5589
Mon Oct 14 10:47:41 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
0/30], Train MSE: 90.7716, Average batch train mse: 90.9088, Val MSE: 236.1309
Step difference of Loss:-21.7130
Mon Oct 14 10:48:26 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
11/30], Train MSE: 79.5408, Average batch train mse: 79.7537, Val MSE: 235.4845
Step difference of Loss:0.6464
Mon Oct 14 10:49:11 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
12/30], Train MSE: 73.2569, Average batch train mse: 73.3342, Val MSE: 231.5933
Step difference of Loss:3.8912
Mon Oct 14 10:49:56 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
13/30], Train MSE: 65.1922, Average batch train mse: 65.1699, Val MSE: 224.7379
Step difference of Loss:6.8553
Mon Oct 14 10:50:40 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
14/30], Train MSE: 61.0462, Average batch train mse: 61.1065, Val MSE: 217.1196
Step difference of Loss:7.6183
Mon Oct 14 10:51:25 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
15/30], Train MSE: 57.3621, Average batch train mse: 57.4022, Val MSE: 227.7523
Step difference of Loss:-10.6327
Mon Oct 14 10:52:10 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
16/30], Train MSE: 55.0039, Average batch train mse: 54.9914, Val MSE: 221.6237
Step difference of Loss:6.1286
Mon Oct 14 10:52:54 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
17/30], Train MSE: 51.5783, Average batch train mse: 51.6239, Val MSE: 213.9407
Step difference of Loss:7.6830
Mon Oct 14 10:53:39 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
18/30], Train MSE: 48.6513, Average batch train mse: 48.7146, Val MSE: 221.2339
Step difference of Loss:-7.2932
Mon Oct 14 10:54:23 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
19/30], Train MSE: 48.8732, Average batch train mse: 48.8644, Val MSE: 225.9095
Step difference of Loss:-4.6756
Mon Oct 14 10:55:08 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
20/30], Train MSE: 45.6898, Average batch train mse: 45.6914, Val MSE: 229.5645
Step difference of Loss:-3.6551
Mon Oct 14 10:55:52 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
21/30], Train MSE: 44.4428, Average batch train mse: 44.5748, Val MSE: 215.5681
Step difference of Loss:13.9965
Mon Oct 14 10:56:36 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
22/30], Train MSE: 42.7972, Average batch train mse: 42.7590, Val MSE: 219.5813
Step difference of Loss:-4.0132
Mon Oct 14 10:57:21 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
23/30], Train MSE: 42.1727, Average batch train mse: 42.2146, Val MSE: 222.0240
Step difference of Loss:-2.4427
Mon Oct 14 10:58:05 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
24/30], Train MSE: 41.0195, Average batch train mse: 41.1151, Val MSE: 218.9211
Step difference of Loss:3.1029
Mon Oct 14 10:58:49 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
25/30], Train MSE: 39.8324, Average batch train mse: 39.8826, Val MSE: 226.7303
Step difference of Loss:-7.8092
Mon Oct 14 10:59:34 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
26/30], Train MSE: 37.4184, Average batch train mse: 37.3882, Val MSE: 218.3071
Step difference of Loss:8.4232
Mon Oct 14 11:00:18 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
27/30], Train MSE: 37.3587, Average batch train mse: 37.3609, Val MSE: 212.7800
Step difference of Loss:5.5271
Mon Oct 14 11:01:02 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
28/30], Train MSE: 37.1318, Average batch train mse: 37.1836, Val MSE: 222.3354
Step difference of Loss:-9.5554
```

In [39]:

```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.0001],
    'batch_size': [128],
    'optimizer': ['Adam', 'AdamW', 'SGD']
}
efficientModel = EfficientNet()
eff_best_model, eff_min_mse, eff_best_params = hyperparameter_tuning_with_grid_search(efficientModel, train_dat
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_51820\4218000815.py:8: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 13:18:48 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
/30], Train MSE: 1427.9711, Average batch train mse: 1428.0357, Val MSE: 1235.5404
Step difference of Loss:inf
Mon Oct 14 13:19:36 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [2
/30], Train MSE: 1110.4940, Average batch train mse: 1109.8845, Val MSE: 979.2443
Step difference of Loss:256.2961
Mon Oct 14 13:20:28 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [3
/30], Train MSE: 814.1625, Average batch train mse: 813.7457, Val MSE: 619.1795
Step difference of Loss:360.0648
Mon Oct 14 13:21:19 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [4
/30], Train MSE: 551.9883, Average batch train mse: 551.6173, Val MSE: 460.2548
Step difference of Loss:158.9247
Mon Oct 14 13:22:11 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [5
/30], Train MSE: 340.1001, Average batch train mse: 340.0684, Val MSE: 275.4634
Step difference of Loss:184.7914
Mon Oct 14 13:23:03 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [6
/30], Train MSE: 207.3795, Average batch train mse: 207.7127, Val MSE: 268.1052
Step difference of Loss:7.3582
Mon Oct 14 13:23:56 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [7
/30], Train MSE: 133.9726, Average batch train mse: 134.0828, Val MSE: 264.8901
Step difference of Loss:3.2151
Mon Oct 14 13:24:49 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [8
/30], Train MSE: 103.5108, Average batch train mse: 103.5647, Val MSE: 270.1110
Step difference of Loss:-5.2209
Mon Oct 14 13:25:41 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [9
/30], Train MSE: 85.9743, Average batch train mse: 85.8779, Val MSE: 217.0603
Step difference of Loss:53.0507
Mon Oct 14 13:26:32 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [1
0/30], Train MSE: 75.3578, Average batch train mse: 75.4256, Val MSE: 248.1087
Step difference of Loss:-31.0485
Mon Oct 14 13:27:24 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
11/30], Train MSE: 70.4622, Average batch train mse: 70.3967, Val MSE: 237.1105
Step difference of Loss:10.9983
Mon Oct 14 13:28:11 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
12/30], Train MSE: 66.0641, Average batch train mse: 66.1423, Val MSE: 239.7717
Step difference of Loss:-2.6612
Mon Oct 14 13:28:58 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
13/30], Train MSE: 61.0762, Average batch train mse: 61.0799, Val MSE: 238.3053
Step difference of Loss:1.4664
Mon Oct 14 13:29:48 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
14/30], Train MSE: 55.8714, Average batch train mse: 55.8637, Val MSE: 228.6451
Step difference of Loss:9.6602
Mon Oct 14 13:30:39 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
15/30], Train MSE: 54.5382, Average batch train mse: 54.5862, Val MSE: 222.9490
Step difference of Loss:5.6961
Mon Oct 14 13:31:30 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
16/30], Train MSE: 50.8988, Average batch train mse: 50.9934, Val MSE: 230.1764
Step difference of Loss:-7.2274
Mon Oct 14 13:32:20 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
17/30], Train MSE: 47.5939, Average batch train mse: 47.6345, Val MSE: 223.4247
Step difference of Loss:6.7517
Mon Oct 14 13:33:10 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
18/30], Train MSE: 46.9465, Average batch train mse: 47.0156, Val MSE: 234.5147
Step difference of Loss:-11.0900
Mon Oct 14 13:34:00 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
19/30], Train MSE: 45.0490, Average batch train mse: 45.0737, Val MSE: 223.7846
Step difference of Loss:10.7301
Mon Oct 14 13:34:51 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
20/30], Train MSE: 43.5340, Average batch train mse: 43.5361, Val MSE: 217.4265
Step difference of Loss:6.3581
Mon Oct 14 13:35:41 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
21/30], Train MSE: 42.4029, Average batch train mse: 42.3918, Val MSE: 229.4065
Step difference of Loss:-11.9799
Mon Oct 14 13:36:34 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
22/30], Train MSE: 41.6053, Average batch train mse: 41.6771, Val MSE: 214.4066
Step difference of Loss:14.9999
Mon Oct 14 13:37:25 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
23/30], Train MSE: 41.2764, Average batch train mse: 41.2274, Val MSE: 215.2376
Step difference of Loss:-0.8310
Mon Oct 14 13:38:19 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
24/30], Train MSE: 38.5231, Average batch train mse: 38.4943, Val MSE: 212.7234
Step difference of Loss:2.5142
Mon Oct 14 13:39:11 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
```

```
25/30], Train MSE: 37.8470, Average batch train mse: 37.8626, Val MSE: 223.5327
Step difference of Loss:-10.8093
Mon Oct 14 13:40:01 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
26/30], Train MSE: 38.1423, Average batch train mse: 38.1675, Val MSE: 219.6423
Step difference of Loss:3.8903
Mon Oct 14 13:40:52 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
27/30], Train MSE: 34.9092, Average batch train mse: 34.9349, Val MSE: 221.4096
Step difference of Loss:-1.7673
Mon Oct 14 13:41:43 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
28/30], Train MSE: 35.7147, Average batch train mse: 35.7148, Val MSE: 216.5988
Step difference of Loss:4.8109
Mon Oct 14 13:42:32 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
29/30], Train MSE: 34.9781, Average batch train mse: 34.9548, Val MSE: 210.7234
Step difference of Loss:5.8754
Mon Oct 14 13:43:23 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'Adam'}, Epoch [
30/30], Train MSE: 36.1756, Average batch train mse: 36.1336, Val MSE: 219.6064
Step difference of Loss:-8.8830
Mon Oct 14 13:44:12 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
1/30], Train MSE: 1426.3640, Average batch train mse: 1425.6886, Val MSE: 1216.0903
Step difference of Loss:inf
Mon Oct 14 13:45:00 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
2/30], Train MSE: 1115.8929, Average batch train mse: 1115.6104, Val MSE: 883.8492
Step difference of Loss:332.2411
Mon Oct 14 13:45:47 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
3/30], Train MSE: 821.6817, Average batch train mse: 821.0507, Val MSE: 733.6517
Step difference of Loss:150.1976
Mon Oct 14 13:46:33 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
4/30], Train MSE: 559.8941, Average batch train mse: 559.6200, Val MSE: 507.5209
Step difference of Loss:226.1308
Mon Oct 14 13:47:20 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
5/30], Train MSE: 355.6925, Average batch train mse: 355.5388, Val MSE: 321.4602
Step difference of Loss:186.0607
Mon Oct 14 13:48:06 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
6/30], Train MSE: 220.8515, Average batch train mse: 221.0335, Val MSE: 291.9516
Step difference of Loss:29.5086
Mon Oct 14 13:48:55 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
7/30], Train MSE: 147.8165, Average batch train mse: 147.8246, Val MSE: 304.2130
Step difference of Loss:-12.2614
Mon Oct 14 13:49:45 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
8/30], Train MSE: 110.7100, Average batch train mse: 110.8136, Val MSE: 316.1736
Step difference of Loss:-11.9606
Mon Oct 14 13:50:36 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
9/30], Train MSE: 93.8974, Average batch train mse: 93.8935, Val MSE: 279.8393
Step difference of Loss:36.3344
Mon Oct 14 13:51:26 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [
10/30], Train MSE: 83.1170, Average batch train mse: 83.0373, Val MSE: 240.0759
Step difference of Loss:39.7634
Mon Oct 14 13:52:18 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[11/30], Train MSE: 73.5673, Average batch train mse: 73.5716, Val MSE: 292.0815
Step difference of Loss:-52.0056
Mon Oct 14 13:53:11 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[12/30], Train MSE: 67.6413, Average batch train mse: 67.6233, Val MSE: 246.0754
Step difference of Loss:46.0060
Mon Oct 14 13:54:02 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[13/30], Train MSE: 64.6004, Average batch train mse: 64.6377, Val MSE: 279.8659
Step difference of Loss:-33.7905
Mon Oct 14 13:54:52 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[14/30], Train MSE: 58.6343, Average batch train mse: 58.6075, Val MSE: 267.1454
Step difference of Loss:12.7205
Mon Oct 14 13:55:42 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[15/30], Train MSE: 57.1484, Average batch train mse: 57.1698, Val MSE: 235.9893
Step difference of Loss:31.1562
Mon Oct 14 13:56:33 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[16/30], Train MSE: 51.5976, Average batch train mse: 51.6906, Val MSE: 247.2537
Step difference of Loss:-11.2645
Mon Oct 14 13:57:24 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[17/30], Train MSE: 48.4823, Average batch train mse: 48.5293, Val MSE: 235.8056
Step difference of Loss:11.4482
Mon Oct 14 13:58:11 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[18/30], Train MSE: 47.3358, Average batch train mse: 47.3715, Val MSE: 226.8787
Step difference of Loss:8.9269
Mon Oct 14 13:58:58 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[19/30], Train MSE: 45.0166, Average batch train mse: 44.9694, Val MSE: 229.0697
Step difference of Loss:-2.1910
Mon Oct 14 13:59:44 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[20/30], Train MSE: 45.3480, Average batch train mse: 45.3398, Val MSE: 236.6388
Step difference of Loss:-7.5691
Mon Oct 14 14:00:29 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[21/30], Train MSE: 41.5114, Average batch train mse: 41.5252, Val MSE: 233.8037
Step difference of Loss:2.8351
Mon Oct 14 14:01:15 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch
[22/30], Train MSE: 40.5726, Average batch train mse: 40.5705, Val MSE: 228.3685
Step difference of Loss:5.4353
```

```
Mon Oct 14 14:02:01 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [23/30], Train MSE: 38.9289, Average batch train mse: 38.9353, Val MSE: 227.4451
Step difference of Loss:0.9234
Mon Oct 14 14:02:47 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [24/30], Train MSE: 38.4679, Average batch train mse: 38.4895, Val MSE: 214.4928
Step difference of Loss:12.9524
Mon Oct 14 14:03:33 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [25/30], Train MSE: 37.1346, Average batch train mse: 37.1231, Val MSE: 229.4776
Step difference of Loss:-14.9848
Mon Oct 14 14:04:18 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [26/30], Train MSE: 36.6032, Average batch train mse: 36.6210, Val MSE: 244.2924
Step difference of Loss:-14.8148
Mon Oct 14 14:05:03 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [27/30], Train MSE: 36.3562, Average batch train mse: 36.3745, Val MSE: 222.1668
Step difference of Loss:22.1256
Mon Oct 14 14:05:49 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [28/30], Train MSE: 35.2909, Average batch train mse: 35.2460, Val MSE: 233.0738
Step difference of Loss:-10.9070
Mon Oct 14 14:06:34 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [29/30], Train MSE: 34.6134, Average batch train mse: 34.5895, Val MSE: 226.7950
Step difference of Loss:6.2788
Mon Oct 14 14:07:20 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}, Epoch [30/30], Train MSE: 32.6867, Average batch train mse: 32.6870, Val MSE: 225.5074
Step difference of Loss:1.2876
Mon Oct 14 14:08:07 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [1/30], Train MSE: 704.9617, Average batch train mse: 703.7552, Val MSE: 7108.1763
Step difference of Loss:inf
Mon Oct 14 14:08:54 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2/30], Train MSE: 226.0154, Average batch train mse: 225.9811, Val MSE: 278.7688
Step difference of Loss:6829.4077
Mon Oct 14 14:09:40 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [3/30], Train MSE: 188.0616, Average batch train mse: 188.0917, Val MSE: 182.0933
Step difference of Loss:96.6755
Mon Oct 14 14:10:26 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [4/30], Train MSE: 170.4533, Average batch train mse: 170.3540, Val MSE: 192.2243
Step difference of Loss:-10.1310
Mon Oct 14 14:11:13 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [5/30], Train MSE: 157.8680, Average batch train mse: 157.8665, Val MSE: 198.5215
Step difference of Loss:-6.2972
Mon Oct 14 14:12:07 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [6/30], Train MSE: 147.3906, Average batch train mse: 147.4402, Val MSE: 177.9037
Step difference of Loss:20.6178
Mon Oct 14 14:12:59 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [7/30], Train MSE: 137.8361, Average batch train mse: 137.8789, Val MSE: 163.6895
Step difference of Loss:14.2142
Mon Oct 14 14:13:51 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [8/30], Train MSE: 128.4958, Average batch train mse: 128.4433, Val MSE: 183.7513
Step difference of Loss:-20.0618
Mon Oct 14 14:14:43 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [9/30], Train MSE: 118.6015, Average batch train mse: 118.7179, Val MSE: 169.0692
Step difference of Loss:14.6821
Mon Oct 14 14:15:34 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [10/30], Train MSE: 108.8801, Average batch train mse: 108.8544, Val MSE: 168.2715
Step difference of Loss:0.7977
Mon Oct 14 14:16:23 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [11/30], Train MSE: 100.3901, Average batch train mse: 100.3289, Val MSE: 187.7279
Step difference of Loss:-19.4564
Mon Oct 14 14:17:14 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [12/30], Train MSE: 92.2516, Average batch train mse: 92.3931, Val MSE: 172.0581
Step difference of Loss:15.6699
Mon Oct 14 14:18:05 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [13/30], Train MSE: 85.4758, Average batch train mse: 85.4845, Val MSE: 187.7401
Step difference of Loss:-15.6820
Mon Oct 14 14:18:56 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [14/30], Train MSE: 76.6980, Average batch train mse: 76.7402, Val MSE: 186.9661
Step difference of Loss:0.7739
Mon Oct 14 14:19:47 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [15/30], Train MSE: 73.6738, Average batch train mse: 73.6849, Val MSE: 174.0643
Step difference of Loss:12.9018
Mon Oct 14 14:20:37 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [16/30], Train MSE: 66.5828, Average batch train mse: 66.5948, Val MSE: 195.8367
Step difference of Loss:-21.7723
Mon Oct 14 14:21:27 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [17/30], Train MSE: 61.6214, Average batch train mse: 61.5832, Val MSE: 180.8517
Step difference of Loss:14.9850
Mon Oct 14 14:22:18 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [18/30], Train MSE: 59.0217, Average batch train mse: 59.0527, Val MSE: 184.3850
Step difference of Loss:-3.5333
Mon Oct 14 14:23:07 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [19/30], Train MSE: 55.4736, Average batch train mse: 55.4655, Val MSE: 181.3812
Step difference of Loss:3.0038
Mon Oct 14 14:23:57 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [20/30], Train MSE: 50.3909, Average batch train mse: 50.3876, Val MSE: 177.9713
```

```
Step difference of Loss:3.4099
Mon Oct 14 14:24:48 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
1/30], Train MSE: 48.5639, Average batch train mse: 48.6075, Val MSE: 178.4216
Step difference of Loss:-0.4503
Mon Oct 14 14:25:39 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
2/30], Train MSE: 44.5898, Average batch train mse: 44.5742, Val MSE: 175.9970
Step difference of Loss:2.4246
Mon Oct 14 14:26:29 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
3/30], Train MSE: 43.8252, Average batch train mse: 43.8087, Val MSE: 175.9255
Step difference of Loss:0.0715
Mon Oct 14 14:27:18 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
4/30], Train MSE: 40.8464, Average batch train mse: 40.8172, Val MSE: 172.6290
Step difference of Loss:3.2966
Mon Oct 14 14:28:08 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
5/30], Train MSE: 39.2532, Average batch train mse: 39.2998, Val MSE: 186.4121
Step difference of Loss:-13.7831
Mon Oct 14 14:28:58 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
6/30], Train MSE: 37.1014, Average batch train mse: 37.1158, Val MSE: 175.9419
Step difference of Loss:10.4702
Mon Oct 14 14:29:48 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
7/30], Train MSE: 35.9275, Average batch train mse: 35.9549, Val MSE: 179.7815
Step difference of Loss:-3.8396
Mon Oct 14 14:30:38 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
8/30], Train MSE: 34.3993, Average batch train mse: 34.3893, Val MSE: 181.4764
Step difference of Loss:-1.6949
Mon Oct 14 14:31:29 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [2
9/30], Train MSE: 34.5177, Average batch train mse: 34.4946, Val MSE: 178.5146
Step difference of Loss:2.9618
Mon Oct 14 14:32:19 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'SGD'}, Epoch [3
0/30], Train MSE: 32.3887, Average batch train mse: 32.3788, Val MSE: 173.4411
Step difference of Loss:5.0735
Min Mse: 133.353
Best Parameters: {}
```

In [51]:
```python
eff_checkpoint = torch.load('efficientModel.pth')
#def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
#    torch.save({
#            'epoch': epoch,
#            'model_state_dict': model.state_dict(),
#            'optimizer_state_dict': optimizer.state_dict(),
#            'best params:': best_params,
#            'best loss': best_loss,
#            'training time': training_time,
#            'last val loss': last_val_loss,
#        }, checkpoint_name + '.pth')
print(eff_checkpoint.keys())
print("Min Mse:", eff_checkpoint['best loss'])
print("Best Parameters:", eff_checkpoint['best params:'])
```

```
dict_keys(['epoch', 'model_state_dict', 'optimizer_state_dict', 'best params:', 'best loss', 'training time', 'l
ast val loss'])
Min Mse: 133.353
Best Parameters: {'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 128, 'optimizer': 'AdamW'}
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_130064\479920456.py:1: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  eff_checkpoint = torch.load('efficientModel.pth')
```

4.2 ResNet18 hyperparameter tuning

In [ ]:
```python
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001, 0.0001],
    'batch_size': [512],
    'optimizer': ['Adam', 'AdamW', 'SGD']
}

resnetModel = ResNet18()
res_best_model, res_min_mse, res_best_params = hyperparameter_tuning_with_grid_search(resnetModel, train_datase

print("Min Mse:", res_min_mse)
print("Best Parameters:", res_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_143636\4218000815.py:8: FutureWarning: You are using `torch.load` w
ith `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is pos
sible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github
.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default
value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpi
ckling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowl
isted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 15:15:49 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1/
30], Train MSE: 803.3097, Average batch train mse: 796.6707, Val MSE: 345.2153
Step difference of Loss:inf
Mon Oct 14 15:16:31 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2/
30], Train MSE: 260.4951, Average batch train mse: 259.9556, Val MSE: 239.1464
Step difference of Loss:106.0689
Mon Oct 14 15:17:17 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [3/
30], Train MSE: 214.0938, Average batch train mse: 213.4411, Val MSE: 288.5740
Step difference of Loss:-49.4276
Mon Oct 14 15:18:02 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [4/
30], Train MSE: 191.9847, Average batch train mse: 191.8285, Val MSE: 228.5962
Step difference of Loss:59.9778
Mon Oct 14 15:18:46 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [5/
30], Train MSE: 177.0257, Average batch train mse: 177.1058, Val MSE: 265.2042
Step difference of Loss:-36.6080
Mon Oct 14 15:19:31 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [6/
30], Train MSE: 168.3380, Average batch train mse: 168.3133, Val MSE: 221.8063
Step difference of Loss:43.3979
Mon Oct 14 15:20:16 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [7/
30], Train MSE: 157.3232, Average batch train mse: 157.2719, Val MSE: 165.5522
Step difference of Loss:56.2541
Mon Oct 14 15:21:00 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [8/
30], Train MSE: 150.2988, Average batch train mse: 150.6582, Val MSE: 171.5442
Step difference of Loss:-5.9920
Mon Oct 14 15:21:45 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [9/
30], Train MSE: 140.2712, Average batch train mse: 140.6897, Val MSE: 164.6144
Step difference of Loss:6.9297
Mon Oct 14 15:22:29 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [10
/30], Train MSE: 131.3768, Average batch train mse: 131.7041, Val MSE: 155.5295
Step difference of Loss:9.0849
Mon Oct 14 15:23:14 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
1/30], Train MSE: 121.9950, Average batch train mse: 122.5384, Val MSE: 162.1512
Step difference of Loss:-6.6216
Mon Oct 14 15:23:58 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
2/30], Train MSE: 115.3515, Average batch train mse: 115.4293, Val MSE: 147.0566
Mon Oct 14 15:23:59 2024 best_params upgrade:", {'epochs': 11, 'learning_rate': 0.001, 'batch_size': 512, 'optim
izer': 'Adam'}, min_mse: 147.056640625
Step difference of Loss:15.0945
Mon Oct 14 15:24:43 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
3/30], Train MSE: 107.3865, Average batch train mse: 107.3030, Val MSE: 161.5549
Step difference of Loss:-14.4983
Mon Oct 14 15:25:28 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
4/30], Train MSE: 96.2668, Average batch train mse: 96.1563, Val MSE: 162.7049
Step difference of Loss:-1.1500
Mon Oct 14 15:26:13 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
5/30], Train MSE: 84.9695, Average batch train mse: 84.9334, Val MSE: 163.8525
Step difference of Loss:-1.1476
Mon Oct 14 15:26:58 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
6/30], Train MSE: 71.3837, Average batch train mse: 71.8911, Val MSE: 167.9503
Step difference of Loss:-4.0978
Mon Oct 14 15:27:43 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
7/30], Train MSE: 60.3211, Average batch train mse: 60.4023, Val MSE: 165.5330
Step difference of Loss:2.4173
Mon Oct 14 15:28:27 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
8/30], Train MSE: 51.3085, Average batch train mse: 51.2151, Val MSE: 150.3637
Step difference of Loss:15.1693
Mon Oct 14 15:29:12 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
9/30], Train MSE: 40.8776, Average batch train mse: 40.7953, Val MSE: 153.0244
Step difference of Loss:-2.6607
Mon Oct 14 15:29:57 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
0/30], Train MSE: 35.0109, Average batch train mse: 35.0962, Val MSE: 161.3485
Step difference of Loss:-8.3241
Mon Oct 14 15:30:41 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
1/30], Train MSE: 28.2973, Average batch train mse: 28.2432, Val MSE: 168.0343
Step difference of Loss:-6.6858
Mon Oct 14 15:31:25 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
2/30], Train MSE: 21.9525, Average batch train mse: 22.0598, Val MSE: 153.9258
Step difference of Loss:14.1085
Mon Oct 14 15:32:10 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
3/30], Train MSE: 18.4395, Average batch train mse: 18.7486, Val MSE: 152.4544
Step difference of Loss:1.4715
Mon Oct 14 15:32:55 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
4/30], Train MSE: 15.6829, Average batch train mse: 15.7100, Val MSE: 149.2578
```

```
Step difference of Loss:3.1965
Mon Oct 14 15:33:39 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [25/30], Train MSE: 11.7414, Average batch train mse: 11.7787, Val MSE: 151.0645
Step difference of Loss:-1.8067
Mon Oct 14 15:34:25 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [26/30], Train MSE: 10.1947, Average batch train mse: 10.1474, Val MSE: 155.1973
Step difference of Loss:-4.1328
Mon Oct 14 15:35:10 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [27/30], Train MSE: 8.5279, Average batch train mse: 8.6924, Val MSE: 152.3313
Step difference of Loss:2.8660
Mon Oct 14 15:35:55 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [28/30], Train MSE: 7.3104, Average batch train mse: 7.2807, Val MSE: 148.3638
Step difference of Loss:3.9676
Mon Oct 14 15:36:40 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [29/30], Train MSE: 6.7485, Average batch train mse: 6.7420, Val MSE: 151.3565
Step difference of Loss:-2.9927
Mon Oct 14 15:37:25 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [30/30], Train MSE: 6.0624, Average batch train mse: 6.1731, Val MSE: 149.1646
Step difference of Loss:2.1919
Mon Oct 14 15:38:09 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [1/30], Train MSE: 802.8583, Average batch train mse: 797.3746, Val MSE: 391.4590
Step difference of Loss:inf
Mon Oct 14 15:38:54 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [2/30], Train MSE: 266.8401, Average batch train mse: 266.4079, Val MSE: 240.8890
Step difference of Loss:150.5701
Mon Oct 14 15:39:38 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [3/30], Train MSE: 209.9592, Average batch train mse: 209.9425, Val MSE: 205.1871
Step difference of Loss:35.7019
Mon Oct 14 15:40:23 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [4/30], Train MSE: 191.7300, Average batch train mse: 191.3788, Val MSE: 227.6140
Step difference of Loss:-22.4269
Mon Oct 14 15:41:07 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [5/30], Train MSE: 177.3089, Average batch train mse: 176.9937, Val MSE: 221.4548
Step difference of Loss:6.1591
Mon Oct 14 15:41:51 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [6/30], Train MSE: 169.2063, Average batch train mse: 169.1917, Val MSE: 183.1532
Step difference of Loss:38.3017
Mon Oct 14 15:42:36 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [7/30], Train MSE: 160.6735, Average batch train mse: 160.1204, Val MSE: 222.9977
Step difference of Loss:-39.8445
Mon Oct 14 15:43:21 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [8/30], Train MSE: 152.1023, Average batch train mse: 152.0474, Val MSE: 199.0018
Step difference of Loss:23.9959
Mon Oct 14 15:44:05 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [9/30], Train MSE: 140.3156, Average batch train mse: 140.4932, Val MSE: 196.6293
Step difference of Loss:2.3725
Mon Oct 14 15:44:50 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [10/30], Train MSE: 131.6847, Average batch train mse: 131.7140, Val MSE: 167.4197
Step difference of Loss:29.2096
```

In [100…

```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001],
    'batch_size': [512],
    'optimizer': ['AdamW', 'SGD']
}

resnetModel = ResNet18()
res_best_model, res_min_mse, res_best_params = hyperparameter_tuning_with_grid_search(resnetModel, train_datase

print("Min Mse:", res_min_mse)
print("Best Parameters:", res_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_8996\4218000815.py:8: FutureWarning: You are using `torch.load` wit
h `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possi
ble to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.c
om/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default va
lue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpick
ling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlis
ted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` f
or any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any iss
ues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 16:20:35 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [1/30], Train MSE: 794.1512, Average batch train mse: 787.3090, Val MSE: 298.6559
Step difference of Loss:inf
Mon Oct 14 16:21:13 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [2/30], Train MSE: 258.6104, Average batch train mse: 258.4057, Val MSE: 259.4972
Step difference of Loss:39.1588
Mon Oct 14 16:21:50 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [3/30], Train MSE: 206.0701, Average batch train mse: 206.0234, Val MSE: 212.5158
Step difference of Loss:46.9814
Mon Oct 14 16:22:28 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [4/30], Train MSE: 189.4752, Average batch train mse: 189.3819, Val MSE: 186.1862
```

```
Step difference of Loss:26.3296
Mon Oct 14 16:23:05 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [5
/30], Train MSE: 177.3996, Average batch train mse: 177.6642, Val MSE: 186.2833
Step difference of Loss:-0.0971
Mon Oct 14 16:23:43 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [6
/30], Train MSE: 165.8585, Average batch train mse: 165.3953, Val MSE: 230.7398
Step difference of Loss:-44.4565
Mon Oct 14 16:24:21 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [7
/30], Train MSE: 158.0574, Average batch train mse: 158.8566, Val MSE: 177.1636
Step difference of Loss:53.5762
Mon Oct 14 16:25:00 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [8
/30], Train MSE: 151.8619, Average batch train mse: 152.1588, Val MSE: 188.8454
Step difference of Loss:-11.6817
Mon Oct 14 16:25:39 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [9
/30], Train MSE: 143.7904, Average batch train mse: 143.8024, Val MSE: 160.8523
Step difference of Loss:27.9931
Mon Oct 14 16:26:17 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [1
0/30], Train MSE: 138.6429, Average batch train mse: 138.3401, Val MSE: 171.8318
Step difference of Loss:-10.9795
Mon Oct 14 16:26:55 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
11/30], Train MSE: 128.2671, Average batch train mse: 128.3854, Val MSE: 149.5658
Step difference of Loss:22.2661
Mon Oct 14 16:27:33 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
12/30], Train MSE: 118.0389, Average batch train mse: 118.5049, Val MSE: 154.5009
Step difference of Loss:-4.9351
Mon Oct 14 16:28:12 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
13/30], Train MSE: 108.7568, Average batch train mse: 108.3212, Val MSE: 150.7665
Step difference of Loss:3.7343
Mon Oct 14 16:28:50 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
14/30], Train MSE: 102.6378, Average batch train mse: 102.4909, Val MSE: 172.9629
Step difference of Loss:-22.1964
Mon Oct 14 16:29:28 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
15/30], Train MSE: 86.2343, Average batch train mse: 86.2147, Val MSE: 161.2961
Step difference of Loss:11.6668
Mon Oct 14 16:30:07 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
16/30], Train MSE: 76.1085, Average batch train mse: 76.2479, Val MSE: 155.2491
Step difference of Loss:6.0470
Mon Oct 14 16:30:46 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
17/30], Train MSE: 61.7259, Average batch train mse: 61.9817, Val MSE: 175.1987
Step difference of Loss:-19.9496
Mon Oct 14 16:31:24 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
18/30], Train MSE: 51.3452, Average batch train mse: 51.3658, Val MSE: 170.9209
Step difference of Loss:4.2778
Mon Oct 14 16:32:03 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
19/30], Train MSE: 43.4060, Average batch train mse: 43.2390, Val MSE: 160.7317
Step difference of Loss:10.1892
Mon Oct 14 16:32:41 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
20/30], Train MSE: 34.0974, Average batch train mse: 34.0375, Val MSE: 168.1907
Step difference of Loss:-7.4590
Mon Oct 14 16:33:19 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
21/30], Train MSE: 25.9161, Average batch train mse: 25.8576, Val MSE: 153.3130
Step difference of Loss:14.8777
Mon Oct 14 16:33:57 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
22/30], Train MSE: 22.6402, Average batch train mse: 22.7824, Val MSE: 165.8096
Step difference of Loss:-12.4966
Mon Oct 14 16:34:35 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
23/30], Train MSE: 20.5925, Average batch train mse: 20.6394, Val MSE: 156.2470
Step difference of Loss:9.5626
Mon Oct 14 16:35:13 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
24/30], Train MSE: 16.7392, Average batch train mse: 16.6759, Val MSE: 162.0652
Step difference of Loss:-5.8182
Mon Oct 14 16:35:51 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
25/30], Train MSE: 13.8756, Average batch train mse: 13.8306, Val MSE: 161.9418
Step difference of Loss:0.1234
Mon Oct 14 16:36:29 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
26/30], Train MSE: 11.7534, Average batch train mse: 11.7318, Val MSE: 155.4178
Step difference of Loss:6.5240
Mon Oct 14 16:37:08 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
27/30], Train MSE: 9.7436, Average batch train mse: 9.7291, Val MSE: 154.6200
Step difference of Loss:0.7978
Mon Oct 14 16:37:46 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
28/30], Train MSE: 9.2806, Average batch train mse: 9.3741, Val MSE: 154.9506
Step difference of Loss:-0.3306
Mon Oct 14 16:38:24 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
29/30], Train MSE: 8.4252, Average batch train mse: 8.4618, Val MSE: 150.3022
Step difference of Loss:4.6484
Mon Oct 14 16:39:02 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
30/30], Train MSE: 6.5420, Average batch train mse: 6.5516, Val MSE: 152.6671
Step difference of Loss:-2.3649
Mon Oct 14 16:39:41 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [1/3
0], Train MSE: 292.9538, Average batch train mse: 292.4632, Val MSE: 270.6857
Step difference of Loss:inf
Mon Oct 14 16:40:19 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [2/3
```

0], Train MSE: 219.6844, Average batch train mse: 220.4827, Val MSE: 279.1656
Step difference of Loss:-8.4799
Mon Oct 14 16:40:58 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [3/3
0], Train MSE: 204.4086, Average batch train mse: 203.9173, Val MSE: 214.8603
Step difference of Loss:64.3054
Mon Oct 14 16:41:36 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [4/3
0], Train MSE: 191.5131, Average batch train mse: 190.9601, Val MSE: 196.1857
Step difference of Loss:18.6745
Mon Oct 14 16:42:14 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [5/3
0], Train MSE: 178.1925, Average batch train mse: 178.4327, Val MSE: 202.1775
Step difference of Loss:-5.9918
Mon Oct 14 16:42:52 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [6/3
0], Train MSE: 166.1752, Average batch train mse: 166.6606, Val MSE: 203.9236
Step difference of Loss:-1.7460
Mon Oct 14 16:43:30 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [7/3
0], Train MSE: 153.7911, Average batch train mse: 154.5592, Val MSE: 235.4560
Step difference of Loss:-31.5324
Mon Oct 14 16:44:08 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [8/3
0], Train MSE: 141.9108, Average batch train mse: 141.8314, Val MSE: 224.1648
Step difference of Loss:11.2912
Mon Oct 14 16:44:47 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [9/3
0], Train MSE: 122.4202, Average batch train mse: 122.2114, Val MSE: 187.1713
Step difference of Loss:36.9935
Mon Oct 14 16:45:24 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [10/
30], Train MSE: 106.2160, Average batch train mse: 106.0291, Val MSE: 199.8926
Step difference of Loss:-12.7213
Mon Oct 14 16:46:03 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [11
/30], Train MSE: 81.0602, Average batch train mse: 81.0366, Val MSE: 528.7487
Step difference of Loss:-328.8561
Mon Oct 14 16:46:42 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [12
/30], Train MSE: 61.0302, Average batch train mse: 61.1153, Val MSE: 242.3460
Step difference of Loss:286.4027
Mon Oct 14 16:47:20 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [13
/30], Train MSE: 44.8450, Average batch train mse: 45.0165, Val MSE: 304.5022
Step difference of Loss:-62.1562
Mon Oct 14 16:47:59 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [14
/30], Train MSE: 32.2622, Average batch train mse: 32.2781, Val MSE: 186.9224
Step difference of Loss:117.5798
Mon Oct 14 16:48:37 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [15
/30], Train MSE: 20.9491, Average batch train mse: 20.9791, Val MSE: 192.0158
Step difference of Loss:-5.0934
Mon Oct 14 16:49:16 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [16
/30], Train MSE: 16.4511, Average batch train mse: 16.4205, Val MSE: 188.4760
Step difference of Loss:3.5398
Mon Oct 14 16:49:54 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [17
/30], Train MSE: 11.4933, Average batch train mse: 11.5052, Val MSE: 213.5464
Step difference of Loss:-25.0704
Mon Oct 14 16:50:31 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [18
/30], Train MSE: 9.5490, Average batch train mse: 9.5205, Val MSE: 179.6463
Step difference of Loss:33.9001
Mon Oct 14 16:51:08 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [19
/30], Train MSE: 7.3856, Average batch train mse: 7.3643, Val MSE: 177.0074
Step difference of Loss:2.6389
Mon Oct 14 16:51:46 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [20
/30], Train MSE: 6.1134, Average batch train mse: 6.1534, Val MSE: 186.6452
Step difference of Loss:-9.6378
Mon Oct 14 16:52:25 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [21
/30], Train MSE: 5.5083, Average batch train mse: 5.4833, Val MSE: 176.8016
Step difference of Loss:9.8436
Mon Oct 14 16:53:03 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [22
/30], Train MSE: 5.3578, Average batch train mse: 5.4998, Val MSE: 179.4378
Step difference of Loss:-2.6362
Mon Oct 14 16:53:41 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [23
/30], Train MSE: 4.2603, Average batch train mse: 4.2358, Val MSE: 175.6522
Step difference of Loss:3.7855
Mon Oct 14 16:54:19 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [24
/30], Train MSE: 4.2063, Average batch train mse: 4.2375, Val MSE: 182.7637
Step difference of Loss:-7.1115
Mon Oct 14 16:54:57 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [25
/30], Train MSE: 3.6765, Average batch train mse: 3.6837, Val MSE: 180.5643
Step difference of Loss:2.1994
Mon Oct 14 16:55:36 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [26
/30], Train MSE: 3.0446, Average batch train mse: 3.1061, Val MSE: 176.7746
Step difference of Loss:3.7898
Mon Oct 14 16:56:14 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [27
/30], Train MSE: 3.6408, Average batch train mse: 3.7781, Val MSE: 177.7433
Step difference of Loss:-0.9688
Mon Oct 14 16:56:52 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [28
/30], Train MSE: 3.4702, Average batch train mse: 3.4365, Val MSE: 175.5391
Step difference of Loss:2.2043
Mon Oct 14 16:57:31 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [29
/30], Train MSE: 2.8528, Average batch train mse: 2.8813, Val MSE: 174.1603
Step difference of Loss:1.3788

```
Mon Oct 14 16:58:09 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [30
/30], Train MSE: 2.9818, Average batch train mse: 2.9780, Val MSE: 174.6245
Step difference of Loss:-0.4643
Min Mse: 147.05664
Best Parameters: {}
```

In [101…

```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.0001],
    'batch_size': [512],
    'optimizer': ['Adam', 'AdamW', 'SGD']
}

resnetModel = ResNet18()
res_best_model, res_min_mse, res_best_params = hyperparameter_tuning_with_grid_search(resnetModel, train_datase

print("Min Mse:", res_min_mse)
print("Best Parameters:", res_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_8996\4218000815.py:8: FutureWarning: You are using `torch.load` wit
h `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possi
ble to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.c
om/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default va
lue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpick
ling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlis
ted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` f
or any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any iss
ues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 16:58:48 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
/30], Train MSE: 1281.5868, Average batch train mse: 1278.5708, Val MSE: 1152.8807
Step difference of Loss:inf
Mon Oct 14 16:59:26 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [2
/30], Train MSE: 1001.5314, Average batch train mse: 1001.6056, Val MSE: 928.4955
Step difference of Loss:224.3853
Mon Oct 14 17:00:04 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [3
/30], Train MSE: 848.5563, Average batch train mse: 847.3394, Val MSE: 748.3782
Step difference of Loss:180.1172
Mon Oct 14 17:00:43 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [4
/30], Train MSE: 735.8058, Average batch train mse: 735.4596, Val MSE: 686.6924
Step difference of Loss:61.6859
Mon Oct 14 17:01:21 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [5
/30], Train MSE: 634.3677, Average batch train mse: 634.2293, Val MSE: 686.8923
Step difference of Loss:-0.1999
Mon Oct 14 17:01:59 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [6
/30], Train MSE: 541.9293, Average batch train mse: 542.1349, Val MSE: 584.3907
Step difference of Loss:102.5016
Mon Oct 14 17:02:37 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [7
/30], Train MSE: 464.4354, Average batch train mse: 463.2843, Val MSE: 407.3055
Step difference of Loss:177.0852
Mon Oct 14 17:03:15 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [8
/30], Train MSE: 396.5484, Average batch train mse: 395.8145, Val MSE: 331.2668
Step difference of Loss:76.0387
Mon Oct 14 17:03:53 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [9
/30], Train MSE: 336.7499, Average batch train mse: 336.6267, Val MSE: 325.6934
Step difference of Loss:5.5734
Mon Oct 14 17:04:32 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [1
0/30], Train MSE: 290.9711, Average batch train mse: 290.3868, Val MSE: 263.5200
Step difference of Loss:62.1733
Mon Oct 14 17:05:10 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
11/30], Train MSE: 250.2796, Average batch train mse: 250.4751, Val MSE: 280.3167
Step difference of Loss:-16.7966
Mon Oct 14 17:05:48 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
12/30], Train MSE: 219.0201, Average batch train mse: 218.4886, Val MSE: 197.4503
Step difference of Loss:82.8664
Mon Oct 14 17:06:26 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
13/30], Train MSE: 191.9212, Average batch train mse: 191.1037, Val MSE: 237.7936
Step difference of Loss:-40.3433
Mon Oct 14 17:07:05 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
14/30], Train MSE: 169.8298, Average batch train mse: 169.9733, Val MSE: 183.3989
Step difference of Loss:54.3947
Mon Oct 14 17:07:43 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
15/30], Train MSE: 152.2343, Average batch train mse: 151.8733, Val MSE: 189.3400
Step difference of Loss:-5.9412
Mon Oct 14 17:08:21 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
16/30], Train MSE: 135.0989, Average batch train mse: 135.2570, Val MSE: 177.9880
Step difference of Loss:11.3521
Mon Oct 14 17:09:00 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
17/30], Train MSE: 116.2616, Average batch train mse: 116.2013, Val MSE: 193.3766
Step difference of Loss:-15.3886
Mon Oct 14 17:09:38 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
18/30], Train MSE: 98.9820, Average batch train mse: 98.8165, Val MSE: 331.3052
Step difference of Loss:-137.9286
Mon Oct 14 17:10:16 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
```

```
19/30], Train MSE: 86.0734, Average batch train mse: 86.0851, Val MSE: 184.1493
Step difference of Loss:147.1559
Mon Oct 14 17:10:55 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
20/30], Train MSE: 73.1100, Average batch train mse: 72.9761, Val MSE: 344.2670
Step difference of Loss:-160.1177
Mon Oct 14 17:11:33 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
21/30], Train MSE: 60.5528, Average batch train mse: 60.4811, Val MSE: 192.1323
Step difference of Loss:152.1347
Mon Oct 14 17:12:11 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
22/30], Train MSE: 49.6639, Average batch train mse: 49.8360, Val MSE: 186.6636
Step difference of Loss:5.4687
Mon Oct 14 17:12:49 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
23/30], Train MSE: 41.4443, Average batch train mse: 41.4296, Val MSE: 191.9517
Step difference of Loss:-5.2881
Mon Oct 14 17:13:27 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
24/30], Train MSE: 35.7799, Average batch train mse: 35.7098, Val MSE: 189.7207
Step difference of Loss:2.2310
Mon Oct 14 17:14:05 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
25/30], Train MSE: 28.6164, Average batch train mse: 28.6407, Val MSE: 185.2896
Step difference of Loss:4.4311
Mon Oct 14 17:14:43 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
26/30], Train MSE: 23.5016, Average batch train mse: 23.4849, Val MSE: 206.1857
Step difference of Loss:-20.8961
Mon Oct 14 17:15:24 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
27/30], Train MSE: 20.0586, Average batch train mse: 20.2340, Val MSE: 218.3160
Step difference of Loss:-12.1303
Mon Oct 14 17:16:08 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
28/30], Train MSE: 17.1851, Average batch train mse: 17.6138, Val MSE: 207.8202
Step difference of Loss:10.4958
Mon Oct 14 17:16:53 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
29/30], Train MSE: 17.2715, Average batch train mse: 17.2905, Val MSE: 197.4970
Step difference of Loss:10.3232
Mon Oct 14 17:17:37 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'Adam'}, Epoch [
30/30], Train MSE: 13.8973, Average batch train mse: 13.8822, Val MSE: 198.6728
Step difference of Loss:-1.1758
Mon Oct 14 17:18:21 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
1/30], Train MSE: 1308.4602, Average batch train mse: 1305.2389, Val MSE: 1227.6411
Step difference of Loss:inf
Mon Oct 14 17:19:06 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
2/30], Train MSE: 1018.4649, Average batch train mse: 1018.6397, Val MSE: 981.9169
Step difference of Loss:245.7242
Mon Oct 14 17:19:50 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
3/30], Train MSE: 869.9197, Average batch train mse: 869.5552, Val MSE: 720.6202
Step difference of Loss:261.2966
Mon Oct 14 17:20:35 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
4/30], Train MSE: 755.9583, Average batch train mse: 755.0210, Val MSE: 715.2164
Step difference of Loss:5.4039
Mon Oct 14 17:21:19 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
5/30], Train MSE: 651.6118, Average batch train mse: 650.1950, Val MSE: 622.6384
Step difference of Loss:92.5779
Mon Oct 14 17:22:04 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
6/30], Train MSE: 559.3483, Average batch train mse: 557.8583, Val MSE: 637.2330
Step difference of Loss:-14.5945
Mon Oct 14 17:22:49 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
7/30], Train MSE: 481.4928, Average batch train mse: 481.2391, Val MSE: 397.4473
Step difference of Loss:239.7856
Mon Oct 14 17:23:33 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
8/30], Train MSE: 414.0617, Average batch train mse: 412.8622, Val MSE: 546.8299
Step difference of Loss:-149.3826
Mon Oct 14 17:24:18 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
9/30], Train MSE: 357.9065, Average batch train mse: 358.4099, Val MSE: 323.9305
Step difference of Loss:222.8994
Mon Oct 14 17:25:03 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [
10/30], Train MSE: 306.1028, Average batch train mse: 304.9339, Val MSE: 271.2130
Step difference of Loss:52.7174
Mon Oct 14 17:25:48 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[11/30], Train MSE: 266.8792, Average batch train mse: 265.8034, Val MSE: 341.9795
Step difference of Loss:-70.7665
Mon Oct 14 17:26:32 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[12/30], Train MSE: 227.6096, Average batch train mse: 226.4839, Val MSE: 266.6083
Step difference of Loss:75.3712
Mon Oct 14 17:27:17 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[13/30], Train MSE: 199.8922, Average batch train mse: 199.9774, Val MSE: 203.2957
Step difference of Loss:63.3126
Mon Oct 14 17:28:01 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[14/30], Train MSE: 176.6003, Average batch train mse: 176.6729, Val MSE: 199.2155
Step difference of Loss:4.0802
Mon Oct 14 17:28:46 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[15/30], Train MSE: 160.8569, Average batch train mse: 160.7037, Val MSE: 189.4761
Step difference of Loss:9.7394
Mon Oct 14 17:29:31 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch
[16/30], Train MSE: 141.2288, Average batch train mse: 141.6097, Val MSE: 175.1267
Step difference of Loss:14.3494
```

```
Mon Oct 14 17:30:16 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [17/30], Train MSE: 126.1473, Average batch train mse: 125.8379, Val MSE: 184.1297
Step difference of Loss:-9.0030
Mon Oct 14 17:31:00 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [18/30], Train MSE: 113.4313, Average batch train mse: 113.0326, Val MSE: 220.2342
Step difference of Loss:-36.1045
Mon Oct 14 17:31:45 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [19/30], Train MSE: 98.9737, Average batch train mse: 98.8358, Val MSE: 286.0020
Step difference of Loss:-65.7678
Mon Oct 14 17:32:30 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [20/30], Train MSE: 87.3011, Average batch train mse: 87.3032, Val MSE: 194.9966
Step difference of Loss:91.0054
Mon Oct 14 17:33:15 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [21/30], Train MSE: 72.1385, Average batch train mse: 72.3334, Val MSE: 187.8712
Step difference of Loss:7.1255
Mon Oct 14 17:34:00 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [22/30], Train MSE: 62.4678, Average batch train mse: 62.5344, Val MSE: 200.2213
Step difference of Loss:-12.3501
Mon Oct 14 17:34:45 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [23/30], Train MSE: 52.6508, Average batch train mse: 52.9027, Val MSE: 196.5437
Step difference of Loss:3.6775
Mon Oct 14 17:35:30 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [24/30], Train MSE: 42.8669, Average batch train mse: 43.1803, Val MSE: 279.8601
Step difference of Loss:-83.3163
Mon Oct 14 17:36:16 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [25/30], Train MSE: 37.1215, Average batch train mse: 37.0770, Val MSE: 223.1282
Step difference of Loss:56.7319
Mon Oct 14 17:37:01 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [26/30], Train MSE: 29.7668, Average batch train mse: 29.7918, Val MSE: 331.1727
Step difference of Loss:-108.0446
Mon Oct 14 17:37:46 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [27/30], Train MSE: 27.4805, Average batch train mse: 27.8029, Val MSE: 200.0895
Step difference of Loss:131.0832
Mon Oct 14 17:38:31 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [28/30], Train MSE: 22.7420, Average batch train mse: 22.6881, Val MSE: 216.4726
Step difference of Loss:-16.3831
Mon Oct 14 17:39:16 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [29/30], Train MSE: 20.2007, Average batch train mse: 20.4403, Val MSE: 299.8319
Step difference of Loss:-83.3593
Mon Oct 14 17:40:01 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'AdamW'}, Epoch [30/30], Train MSE: 16.9130, Average batch train mse: 16.8734, Val MSE: 219.7635
Step difference of Loss:80.0684
Mon Oct 14 17:40:46 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [1/30], Train MSE: 537.5995, Average batch train mse: 532.9383, Val MSE: 277.8300
Step difference of Loss:inf
Mon Oct 14 17:41:30 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [2/30], Train MSE: 250.3091, Average batch train mse: 250.5706, Val MSE: 255.3189
Step difference of Loss:22.5111
Mon Oct 14 17:42:15 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [3/30], Train MSE: 237.8196, Average batch train mse: 238.6753, Val MSE: 262.8557
Step difference of Loss:-7.5368
Mon Oct 14 17:43:00 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [4/30], Train MSE: 227.1852, Average batch train mse: 226.9465, Val MSE: 228.3628
Step difference of Loss:34.4930
Mon Oct 14 17:43:45 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [5/30], Train MSE: 215.8100, Average batch train mse: 216.1576, Val MSE: 231.9526
Step difference of Loss:-3.5898
Mon Oct 14 17:44:29 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [6/30], Train MSE: 208.5027, Average batch train mse: 208.6014, Val MSE: 288.4589
Step difference of Loss:-56.5063
Mon Oct 14 17:45:15 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [7/30], Train MSE: 198.6007, Average batch train mse: 198.4934, Val MSE: 206.3090
Step difference of Loss:82.1499
Mon Oct 14 17:46:00 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [8/30], Train MSE: 194.4771, Average batch train mse: 193.8167, Val MSE: 203.8191
Step difference of Loss:2.4899
Mon Oct 14 17:46:45 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [9/30], Train MSE: 188.0022, Average batch train mse: 187.5201, Val MSE: 231.2559
Step difference of Loss:-27.4368
Mon Oct 14 17:47:31 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [10/30], Train MSE: 181.4829, Average batch train mse: 181.5967, Val MSE: 276.7755
Step difference of Loss:-45.5197
Mon Oct 14 17:48:16 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [11/30], Train MSE: 177.0928, Average batch train mse: 177.1941, Val MSE: 732.4755
Step difference of Loss:-455.6999
Mon Oct 14 17:49:02 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [12/30], Train MSE: 173.3438, Average batch train mse: 172.9447, Val MSE: 211.8591
Step difference of Loss:520.6163
Mon Oct 14 17:49:47 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [13/30], Train MSE: 169.9859, Average batch train mse: 169.8492, Val MSE: 262.0679
Step difference of Loss:-50.2088
Mon Oct 14 17:50:32 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [14/30], Train MSE: 164.4272, Average batch train mse: 164.5846, Val MSE: 180.2383
```

```
Step difference of Loss:81.8297
Mon Oct 14 17:51:18 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [15/30], Train MSE: 160.5555, Average batch train mse: 160.4564, Val MSE: 299.2382
Step difference of Loss:-119.0000
Mon Oct 14 17:52:03 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [16/30], Train MSE: 157.7292, Average batch train mse: 157.7427, Val MSE: 201.6103
Step difference of Loss:97.6279
Mon Oct 14 17:52:50 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [17/30], Train MSE: 153.9571, Average batch train mse: 154.3344, Val MSE: 180.3739
Step difference of Loss:21.2363
Mon Oct 14 17:53:36 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [18/30], Train MSE: 148.8286, Average batch train mse: 148.6188, Val MSE: 220.3996
Step difference of Loss:-40.0257
Mon Oct 14 17:54:22 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [19/30], Train MSE: 145.4440, Average batch train mse: 145.3457, Val MSE: 233.7139
Step difference of Loss:-13.3143
Mon Oct 14 17:55:08 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [20/30], Train MSE: 142.0269, Average batch train mse: 143.2034, Val MSE: 211.2420
Step difference of Loss:22.4719
Mon Oct 14 17:55:54 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [21/30], Train MSE: 140.6522, Average batch train mse: 140.1968, Val MSE: 174.5264
Step difference of Loss:36.7156
Mon Oct 14 17:56:39 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [22/30], Train MSE: 133.8844, Average batch train mse: 134.0613, Val MSE: 177.7909
Step difference of Loss:-3.2645
Mon Oct 14 17:57:26 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [23/30], Train MSE: 128.2052, Average batch train mse: 128.5560, Val MSE: 221.5155
Step difference of Loss:-43.7246
Mon Oct 14 17:58:12 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [24/30], Train MSE: 124.6756, Average batch train mse: 124.7325, Val MSE: 199.7514
Step difference of Loss:21.7641
Mon Oct 14 17:58:57 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [25/30], Train MSE: 117.7698, Average batch train mse: 117.8704, Val MSE: 228.7085
Step difference of Loss:-28.9571
Mon Oct 14 17:59:43 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [26/30], Train MSE: 113.9271, Average batch train mse: 114.6226, Val MSE: 221.4516
Step difference of Loss:7.2569
Mon Oct 14 18:00:29 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [27/30], Train MSE: 104.1533, Average batch train mse: 104.1863, Val MSE: 193.6877
Step difference of Loss:27.7639
Mon Oct 14 18:01:16 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [28/30], Train MSE: 98.0657, Average batch train mse: 98.4395, Val MSE: 342.7856
Step difference of Loss:-149.0979
Mon Oct 14 18:02:02 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [29/30], Train MSE: 93.8566, Average batch train mse: 93.5810, Val MSE: 184.3928
Step difference of Loss:158.3928
Mon Oct 14 18:02:48 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 512, 'optimizer': 'SGD'}, Epoch [30/30], Train MSE: 83.8380, Average batch train mse: 83.9133, Val MSE: 560.7202
Step difference of Loss:-376.3273
Min Mse: 147.05664
Best Parameters: {}
```

In [53]:

```python
res_checkpoint = torch.load('resnetModel.pth')
#def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
#    torch.save({
#            'epoch': epoch,
#            'model_state_dict': model.state_dict(),
#            'optimizer_state_dict': optimizer.state_dict(),
#            'best params:': best_params,
#            'best loss': best_loss,
#            'training time': training_time,
#            'last val loss': last_val_loss,
#        }, checkpoint_name + '.pth')
print(res_checkpoint.keys())
print("Min Mse:", res_checkpoint['best loss'])
print("Best Parameters:", res_checkpoint['best params:'])
```

```
dict_keys(['epoch', 'model_state_dict', 'optimizer_state_dict', 'best params:', 'best loss', 'training time', 'last val loss'])
Min Mse: 147.05664
Best Parameters: {'epochs': 11, 'learning_rate': 0.001, 'batch_size': 512, 'optimizer': 'Adam'}
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_130064\2844795324.py:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  res_checkpoint = torch.load('resnetModel.pth')
```

4.3 Swin hyperparameter tuning

In [39]:
```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001],
    'batch_size': [64],
    'optimizer': ['Adam']
}

swinModel = SwinTransformerWithMLP()
swin_best_model, swin_min_mse, swin_best_params = hyperparameter_tuning_with_grid_search(swinModel, train_datas
print("Min Mse:", swin_min_mse)
print("Best Parameters:", swin_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\4218000815.py:8: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 19:59:09 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [1/3
0], Train MSE: 311.6355, Average batch train mse: 311.8526, Val MSE: 286.8236
Step difference of Loss:inf
Mon Oct 14 20:00:21 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [2/3
0], Train MSE: 266.2554, Average batch train mse: 266.1618, Val MSE: 255.6155
Step difference of Loss:31.2081
Mon Oct 14 20:01:34 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [3/3
0], Train MSE: 247.4138, Average batch train mse: 247.2525, Val MSE: 261.2281
Step difference of Loss:-5.6126
Mon Oct 14 20:02:48 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [4/3
0], Train MSE: 238.2319, Average batch train mse: 239.0555, Val MSE: 238.4897
Step difference of Loss:22.7383
Mon Oct 14 20:04:08 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [5/3
0], Train MSE: 233.0908, Average batch train mse: 233.2271, Val MSE: 231.4888
Step difference of Loss:7.0010
Mon Oct 14 20:05:27 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [6/3
0], Train MSE: 225.0249, Average batch train mse: 224.6794, Val MSE: 224.1147
Step difference of Loss:7.3741
Mon Oct 14 20:06:46 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [7/3
0], Train MSE: 219.0859, Average batch train mse: 218.7574, Val MSE: 218.1710
Step difference of Loss:5.9437
Mon Oct 14 20:08:06 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [8/3
0], Train MSE: 211.8927, Average batch train mse: 211.7747, Val MSE: 219.7118
Step difference of Loss:-1.5408
Mon Oct 14 20:09:25 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [9/3
0], Train MSE: 206.6586, Average batch train mse: 206.7335, Val MSE: 213.4369
Step difference of Loss:6.2749
Mon Oct 14 20:10:46 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [10/
30], Train MSE: 198.6550, Average batch train mse: 198.4853, Val MSE: 214.5755
Step difference of Loss:-1.1386
Mon Oct 14 20:12:07 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [11
/30], Train MSE: 197.2076, Average batch train mse: 196.9800, Val MSE: 209.3630
Step difference of Loss:5.2125
Mon Oct 14 20:13:27 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [12
/30], Train MSE: 190.5845, Average batch train mse: 190.3255, Val MSE: 205.5127
Step difference of Loss:3.8503
Mon Oct 14 20:14:46 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [13
/30], Train MSE: 185.1552, Average batch train mse: 185.4631, Val MSE: 203.1980
Step difference of Loss:2.3147
Mon Oct 14 20:16:04 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [14
/30], Train MSE: 177.5777, Average batch train mse: 177.7334, Val MSE: 209.7525
Step difference of Loss:-6.5544
Mon Oct 14 20:17:23 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [15
/30], Train MSE: 171.8365, Average batch train mse: 171.4495, Val MSE: 230.5690
Step difference of Loss:-20.8166
Mon Oct 14 20:18:42 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [16
/30], Train MSE: 164.6273, Average batch train mse: 164.6109, Val MSE: 216.4546
Step difference of Loss:14.1145
Mon Oct 14 20:20:02 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [17
/30], Train MSE: 156.5439, Average batch train mse: 156.4970, Val MSE: 206.5957
Step difference of Loss:9.8588
Mon Oct 14 20:21:21 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [18
/30], Train MSE: 146.8906, Average batch train mse: 146.9747, Val MSE: 202.6653
Step difference of Loss:3.9304
Mon Oct 14 20:22:41 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [19
/30], Train MSE: 138.2805, Average batch train mse: 138.3799, Val MSE: 222.9658
Step difference of Loss:-20.3005
Mon Oct 14 20:23:59 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [20
/30], Train MSE: 134.0145, Average batch train mse: 134.9500, Val MSE: 251.7478
Step difference of Loss:-28.7819
```

```
Mon Oct 14 20:25:20 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [21
/30], Train MSE: 126.7827, Average batch train mse: 127.0964, Val MSE: 206.1621
Step difference of Loss:45.5856
Mon Oct 14 20:26:39 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [22
/30], Train MSE: 118.8062, Average batch train mse: 118.7469, Val MSE: 211.3575
Step difference of Loss:-5.1954
Mon Oct 14 20:27:58 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [23
/30], Train MSE: 107.2880, Average batch train mse: 107.3534, Val MSE: 220.9546
Step difference of Loss:-9.5971
Mon Oct 14 20:29:18 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [24
/30], Train MSE: 99.4923, Average batch train mse: 99.5457, Val MSE: 225.9064
Step difference of Loss:-4.9518
Mon Oct 14 20:30:37 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [25
/30], Train MSE: 93.9583, Average batch train mse: 93.9272, Val MSE: 220.8751
Step difference of Loss:5.0313
Mon Oct 14 20:31:57 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [26
/30], Train MSE: 85.9246, Average batch train mse: 85.9725, Val MSE: 227.3338
Step difference of Loss:-6.4587
Mon Oct 14 20:33:17 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [27
/30], Train MSE: 80.1346, Average batch train mse: 80.3093, Val MSE: 220.3963
Step difference of Loss:6.9375
Mon Oct 14 20:34:37 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [28
/30], Train MSE: 74.1630, Average batch train mse: 74.3895, Val MSE: 233.8393
Step difference of Loss:-13.4430
Mon Oct 14 20:35:55 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [29
/30], Train MSE: 68.3442, Average batch train mse: 68.2517, Val MSE: 244.0878
Step difference of Loss:-10.2485
Mon Oct 14 20:37:15 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'Adam'}, Epoch [30
/30], Train MSE: 62.1866, Average batch train mse: 62.2709, Val MSE: 234.3164
Step difference of Loss:9.7713
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[39], line 10
      8 swinModel = SwinTransformerWithMLP()
      9 swin_best_model, swin_min_mse, swin_best_params = hyperparameter_tuning_with_grid_search(swinModel, trai
n_dataset, val_dataset, param_grid, 'swinModel', 0.0001)
---> 10 print("Min Mse:", res_min_mse)
     11 print("Best Parameters:", res_best_params)

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

In [42]:
```
swin_checkpoint = torch.load('swinModel.pth')
#def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
#    torch.save({
#            'epoch': epoch,
#            'model_state_dict': model.state_dict(),
#            'optimizer_state_dict': optimizer.state_dict(),
#            'best params:': best_params,
#            'best loss': best_loss,
#            'training time': training_time,
#            'last val loss': last_val_loss,
#        }, checkpoint_name + '.pth')
print(swin_checkpoint.keys())
print("Min Mse:", swin_checkpoint['best loss'])
print("Best Parameters:", swin_checkpoint['best params:'])
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\1766329792.py:1: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  swin_checkpoint = torch.load('swinModel.pth')
dict_keys(['epoch', 'model_state_dict', 'optimizer_state_dict', 'best params:', 'best loss', 'training time', 'l
ast val loss'])
Min Mse: 177.08434
Best Parameters: {'epochs': 19, 'learning_rate': 0.001, 'batch_size': 128, 'optimizer': 'Adam'}
```

In [43]:
```
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001],
    'batch_size': [64],
    'optimizer': ['AdamW']
}

swinModel = SwinTransformerWithMLP()
swin_best_model, swin_min_mse, swin_best_params = hyperparameter_tuning_with_grid_search(swinModel, train_datas
print("Min Mse:", swin_min_mse)
print("Best Parameters:", swin_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\4218000815.py:8: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 20:55:10 2024{'epochs': 0, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1/
30], Train MSE: 303.6395, Average batch train mse: 304.2043, Val MSE: 267.3273
Step difference of Loss:inf
Mon Oct 14 20:56:29 2024{'epochs': 1, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [2/
30], Train MSE: 275.9226, Average batch train mse: 275.7491, Val MSE: 266.6049
Step difference of Loss:0.7224
Mon Oct 14 20:57:44 2024{'epochs': 2, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [3/
30], Train MSE: 272.5556, Average batch train mse: 273.6116, Val MSE: 262.3184
Step difference of Loss:4.2865
Mon Oct 14 20:58:59 2024{'epochs': 3, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [4/
30], Train MSE: 255.9655, Average batch train mse: 256.0214, Val MSE: 245.0079
Step difference of Loss:17.3104
Mon Oct 14 21:00:12 2024{'epochs': 4, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [5/
30], Train MSE: 236.2875, Average batch train mse: 236.3816, Val MSE: 220.5347
Step difference of Loss:24.4733
Mon Oct 14 21:01:29 2024{'epochs': 5, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [6/
30], Train MSE: 218.7368, Average batch train mse: 218.6916, Val MSE: 224.1783
Step difference of Loss:-3.6437
Mon Oct 14 21:02:42 2024{'epochs': 6, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [7/
30], Train MSE: 207.4852, Average batch train mse: 207.7005, Val MSE: 204.1580
Step difference of Loss:20.0203
Mon Oct 14 21:03:56 2024{'epochs': 7, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [8/
30], Train MSE: 203.3961, Average batch train mse: 203.2419, Val MSE: 199.5199
Step difference of Loss:4.6382
Mon Oct 14 21:05:11 2024{'epochs': 8, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [9/
30], Train MSE: 195.7497, Average batch train mse: 195.8054, Val MSE: 217.8852
Step difference of Loss:-18.3653
Mon Oct 14 21:06:24 2024{'epochs': 9, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [10
/30], Train MSE: 186.8976, Average batch train mse: 187.1167, Val MSE: 199.8703
Step difference of Loss:18.0148
Mon Oct 14 21:07:39 2024{'epochs': 10, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
1/30], Train MSE: 185.1023, Average batch train mse: 184.8376, Val MSE: 221.5052
Step difference of Loss:-21.6348
Mon Oct 14 21:08:54 2024{'epochs': 11, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
2/30], Train MSE: 177.4218, Average batch train mse: 177.5234, Val MSE: 187.5584
Step difference of Loss:33.9468
Mon Oct 14 21:10:13 2024{'epochs': 12, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
3/30], Train MSE: 171.7370, Average batch train mse: 172.0015, Val MSE: 181.5694
Step difference of Loss:5.9890
Mon Oct 14 21:11:32 2024{'epochs': 13, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
4/30], Train MSE: 167.5589, Average batch train mse: 167.5156, Val MSE: 179.4642
Step difference of Loss:2.1053
Mon Oct 14 21:12:50 2024{'epochs': 14, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
5/30], Train MSE: 163.6675, Average batch train mse: 163.5911, Val MSE: 175.3056
Mon Oct 14 21:12:51 2024 best_params upgrade:", {'epochs': 14, 'learning_rate': 0.001, 'batch_size': 64, 'optimi
zer': 'AdamW'}, min_mse: 175.30557250976562
Step difference of Loss:4.1586
Mon Oct 14 21:14:10 2024{'epochs': 15, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
6/30], Train MSE: 160.5556, Average batch train mse: 160.3974, Val MSE: 187.5163
Step difference of Loss:-12.2107
Mon Oct 14 21:15:30 2024{'epochs': 16, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
7/30], Train MSE: 161.5594, Average batch train mse: 161.4639, Val MSE: 192.1324
Step difference of Loss:-4.6161
Mon Oct 14 21:16:49 2024{'epochs': 17, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
8/30], Train MSE: 150.7052, Average batch train mse: 150.4900, Val MSE: 183.3800
Step difference of Loss:8.7524
Mon Oct 14 21:18:09 2024{'epochs': 18, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [1
9/30], Train MSE: 145.6613, Average batch train mse: 145.6754, Val MSE: 176.4165
Step difference of Loss:6.9636
Mon Oct 14 21:19:27 2024{'epochs': 19, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [2
0/30], Train MSE: 141.6315, Average batch train mse: 141.6231, Val MSE: 173.9300
Mon Oct 14 21:19:28 2024 best_params upgrade:", {'epochs': 19, 'learning_rate': 0.001, 'batch_size': 64, 'optimi
zer': 'AdamW'}, min_mse: 173.92996215820312
Step difference of Loss:2.4865
Mon Oct 14 21:20:48 2024{'epochs': 20, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [2
1/30], Train MSE: 135.6947, Average batch train mse: 136.1808, Val MSE: 184.8435
Step difference of Loss:-10.9135
Mon Oct 14 21:22:07 2024{'epochs': 21, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [2
2/30], Train MSE: 130.2121, Average batch train mse: 130.4592, Val MSE: 184.0879
Step difference of Loss:0.7556
Mon Oct 14 21:23:27 2024{'epochs': 22, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [2
3/30], Train MSE: 122.5746, Average batch train mse: 122.9047, Val MSE: 179.8954
Step difference of Loss:4.1925
```

```
Mon Oct 14 21:24:47 2024{'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [24/30], Train MSE: 117.8399, Average batch train mse: 117.6851, Val MSE: 173.0199
Mon Oct 14 21:24:47 2024 best_params upgrade:", {'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, min_mse: 173.01986694335938
Step difference of Loss:6.8755
Mon Oct 14 21:26:06 2024{'epochs': 24, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [25/30], Train MSE: 110.1194, Average batch train mse: 110.0499, Val MSE: 174.9186
Step difference of Loss:-1.8987
Mon Oct 14 21:27:25 2024{'epochs': 25, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [26/30], Train MSE: 107.6218, Average batch train mse: 107.5458, Val MSE: 184.8540
Step difference of Loss:-9.9354
Mon Oct 14 21:28:44 2024{'epochs': 26, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [27/30], Train MSE: 98.6760, Average batch train mse: 98.5085, Val MSE: 185.0442
Step difference of Loss:-0.1902
Mon Oct 14 21:30:03 2024{'epochs': 27, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [28/30], Train MSE: 93.1366, Average batch train mse: 93.1946, Val MSE: 209.4855
Step difference of Loss:-24.4413
Mon Oct 14 21:31:22 2024{'epochs': 28, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [29/30], Train MSE: 87.4181, Average batch train mse: 87.3206, Val MSE: 192.5726
Step difference of Loss:16.9129
Mon Oct 14 21:32:41 2024{'epochs': 29, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}, Epoch [30/30], Train MSE: 78.1279, Average batch train mse: 78.1086, Val MSE: 196.7646
Step difference of Loss:-4.1920
Min Mse: 173.01987
Best Parameters: {'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}
```

In [44]:
```python
swin_checkpoint = torch.load('swinModel.pth')
#def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
#    torch.save({
#            'epoch': epoch,
#            'model_state_dict': model.state_dict(),
#            'optimizer_state_dict': optimizer.state_dict(),
#            'best params:': best_params,
#            'best loss': best_loss,
#            'training time': training_time,
#            'last val loss': last_val_loss,
#        }, checkpoint_name + '.pth')
print(swin_checkpoint.keys())
print("Min Mse:", swin_checkpoint['best loss'])
print("Best Parameters:", swin_checkpoint['best params:'])
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\1766329792.py:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  swin_checkpoint = torch.load('swinModel.pth')
dict_keys(['epoch', 'model_state_dict', 'optimizer_state_dict', 'best params:', 'best loss', 'training time', 'last val loss'])
Min Mse: 173.01987
Best Parameters: {'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}
```

In [ ]:
```python
param_grid = {
    'epochs': [30],
    'learning_rate': [0.001],
    'batch_size': [64],
    'optimizer': ['SGD']
}

swinModel = SwinTransformerWithMLP()
swin_best_model, swin_min_mse, swin_best_params = hyperparameter_tuning_with_grid_search(swinModel, train_datas
print("Min Mse:", swin_min_mse)
print("Best Parameters:", swin_best_params)
```

In [93]:
```python
param_grid = {
    'epochs': [30],
    'learning_rate': [0.0001],
    'batch_size': [64],
    'optimizer': ['SGD']
}

swinModel = SwinTransformerWithMLP()
swin_best_model, swin_min_mse, swin_best_params = hyperparameter_tuning_with_grid_search(swinModel, train_datas
print("Min Mse:", swin_min_mse)
print("Best Parameters:", swin_best_params)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\4218000815.py:8: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 22:05:07 2024{'epochs': 0, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [1/3
0], Train MSE: 585.5806, Average batch train mse: 584.7520, Val MSE: 272.1421
Step difference of Loss:inf
Mon Oct 14 22:06:55 2024{'epochs': 1, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [2/3
0], Train MSE: 301.5829, Average batch train mse: 301.4056, Val MSE: 263.0131
Step difference of Loss:9.1290
Mon Oct 14 22:08:45 2024{'epochs': 2, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [3/3
0], Train MSE: 278.1651, Average batch train mse: 278.8433, Val MSE: 398.6009
Step difference of Loss:-135.5878
Mon Oct 14 22:10:33 2024{'epochs': 3, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [4/3
0], Train MSE: 269.9320, Average batch train mse: 269.4746, Val MSE: 343.3336
Step difference of Loss:55.2673
Mon Oct 14 22:12:21 2024{'epochs': 4, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [5/3
0], Train MSE: 268.2276, Average batch train mse: 267.8682, Val MSE: 313.5351
Step difference of Loss:29.7985
Mon Oct 14 22:14:10 2024{'epochs': 5, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [6/3
0], Train MSE: 261.3434, Average batch train mse: 261.1298, Val MSE: 256.4575
Step difference of Loss:57.0776
Mon Oct 14 22:15:58 2024{'epochs': 6, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [7/3
0], Train MSE: 261.0197, Average batch train mse: 261.1311, Val MSE: 258.0577
Step difference of Loss:-1.6002
Mon Oct 14 22:17:46 2024{'epochs': 7, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [8/3
0], Train MSE: 259.4851, Average batch train mse: 259.0875, Val MSE: 353.6394
Step difference of Loss:-95.5816
Mon Oct 14 22:19:32 2024{'epochs': 8, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [9/3
0], Train MSE: 257.9674, Average batch train mse: 257.5295, Val MSE: 260.4034
Step difference of Loss:93.2360
Mon Oct 14 22:21:20 2024{'epochs': 9, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [10/
30], Train MSE: 256.3363, Average batch train mse: 256.7462, Val MSE: 283.8697
Step difference of Loss:-23.4662
Mon Oct 14 22:23:06 2024{'epochs': 10, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [11
/30], Train MSE: 254.2245, Average batch train mse: 254.5520, Val MSE: 308.2028
Step difference of Loss:-24.3332
Mon Oct 14 22:24:53 2024{'epochs': 11, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [12
/30], Train MSE: 250.8393, Average batch train mse: 251.1217, Val MSE: 327.1378
Step difference of Loss:-18.9350
Mon Oct 14 22:26:39 2024{'epochs': 12, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [13
/30], Train MSE: 245.7372, Average batch train mse: 245.8916, Val MSE: 350.5377
Step difference of Loss:-23.3998
Mon Oct 14 22:28:25 2024{'epochs': 13, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [14
/30], Train MSE: 243.3514, Average batch train mse: 243.7955, Val MSE: 451.7111
Step difference of Loss:-101.1734
Mon Oct 14 22:30:15 2024{'epochs': 14, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [15
/30], Train MSE: 239.1365, Average batch train mse: 238.8289, Val MSE: 244.5879
Step difference of Loss:207.1232
Mon Oct 14 22:32:06 2024{'epochs': 15, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [16
/30], Train MSE: 231.3386, Average batch train mse: 231.2989, Val MSE: 261.3056
Step difference of Loss:-16.7177
Mon Oct 14 22:33:55 2024{'epochs': 16, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [17
/30], Train MSE: 226.9975, Average batch train mse: 227.3891, Val MSE: 322.9265
Step difference of Loss:-61.6210
Mon Oct 14 22:35:47 2024{'epochs': 17, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [18
/30], Train MSE: 221.7405, Average batch train mse: 221.7458, Val MSE: 222.6858
Step difference of Loss:100.2408
Mon Oct 14 22:37:36 2024{'epochs': 18, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [19
/30], Train MSE: 216.8284, Average batch train mse: 217.2948, Val MSE: 245.5021
Step difference of Loss:-22.8163
Mon Oct 14 22:39:28 2024{'epochs': 19, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [20
/30], Train MSE: 210.7275, Average batch train mse: 210.8194, Val MSE: 262.6781
Step difference of Loss:-17.1760
Mon Oct 14 22:41:17 2024{'epochs': 20, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [21
/30], Train MSE: 207.9505, Average batch train mse: 208.0353, Val MSE: 232.2597
Step difference of Loss:30.4184
Mon Oct 14 22:43:05 2024{'epochs': 21, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [22
/30], Train MSE: 200.0671, Average batch train mse: 200.3697, Val MSE: 246.0221
Step difference of Loss:-13.7624
Mon Oct 14 22:44:51 2024{'epochs': 22, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [23
/30], Train MSE: 191.9849, Average batch train mse: 192.1181, Val MSE: 394.7016
Step difference of Loss:-148.6795
Mon Oct 14 22:46:37 2024{'epochs': 23, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [24
/30], Train MSE: 183.3227, Average batch train mse: 183.3451, Val MSE: 227.5727
Step difference of Loss:167.1289
Mon Oct 14 22:48:27 2024{'epochs': 24, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [25
```

```
/30], Train MSE: 171.2804, Average batch train mse: 171.3583, Val MSE: 285.8240
Step difference of Loss:-58.2513
Mon Oct 14 22:50:14 2024{'epochs': 25, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [26
/30], Train MSE: 164.3440, Average batch train mse: 164.4552, Val MSE: 277.5339
Step difference of Loss:8.2901
Mon Oct 14 22:52:04 2024{'epochs': 26, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [27
/30], Train MSE: 149.5593, Average batch train mse: 149.4737, Val MSE: 254.9937
Step difference of Loss:22.5403
Mon Oct 14 22:53:52 2024{'epochs': 27, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [28
/30], Train MSE: 134.2287, Average batch train mse: 134.0707, Val MSE: 243.3540
Step difference of Loss:11.6396
Mon Oct 14 22:55:40 2024{'epochs': 28, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [29
/30], Train MSE: 118.3867, Average batch train mse: 118.3327, Val MSE: 255.7166
Step difference of Loss:-12.3626
Mon Oct 14 22:57:26 2024{'epochs': 29, 'learning_rate': 0.0001, 'batch_size': 64, 'optimizer': 'SGD'}, Epoch [30
/30], Train MSE: 102.0791, Average batch train mse: 102.0102, Val MSE: 240.5262
Step difference of Loss:15.1904
Min Mse: 173.01987
Best Parameters: {}
```

In [94]:
```python
swin_checkpoint = torch.load('swinModel.pth')
#def save(epoch, model, optimizer, best_params, best_loss, training_time, last_val_loss, checkpoint_name):
#    torch.save({
#            'epoch': epoch,
#            'model_state_dict': model.state_dict(),
#            'optimizer_state_dict': optimizer.state_dict(),
#            'best params:': best_params,
#            'best loss': best_loss,
#            'training time': training_time,
#            'last val loss': last_val_loss,
#        }, checkpoint_name + '.pth')
print(swin_checkpoint.keys())
print("Min Mse:", swin_checkpoint['best loss'])
print("Best Parameters:", swin_checkpoint['best params:'])
```

```
dict_keys(['epoch', 'model_state_dict', 'optimizer_state_dict', 'best params:', 'best loss', 'training time', 'l
ast val loss'])
Min Mse: 173.01987
Best Parameters: {'epochs': 23, 'learning_rate': 0.001, 'batch_size': 64, 'optimizer': 'AdamW'}
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_87712\1766329792.py:1: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  swin_checkpoint = torch.load('swinModel.pth')
```

5. Evaluation and Comparison

Final Model Evaluation

In [41]:
```python
def test(model, test_dataset, batch_size, device, criterion):
    model.eval()
    test_loss = 0
    all_test_outputs = []
    all_test_labels = []
    test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_worke

    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()

            all_test_outputs.append(outputs.cpu().detach().numpy())
            all_test_labels.append(labels.cpu().detach().numpy())

        average_batch_test_loss = test_loss / len(test_dataloader)
        all_test_outputs = np.concatenate(all_test_outputs)
        all_test_labels = np.concatenate(all_test_labels)

        test_mse = np.mean((all_test_outputs - all_test_labels) ** 2)  # Calculate MSE for the entire test set
        time_str = time.asctime(time.localtime(time.time()))
```

```
        print(f'{time_str}, Test MSE: {test_mse:.4f}, Average batch Test loss: {average_batch_test_loss:.4f}')

    return test_mse, average_batch_test_loss
```

In [52]:
```
def test_mae(model, test_dataset, batch_size, device, criterion):
    model.eval()
    test_loss = 0
    all_test_outputs = []
    all_test_labels = []
    test_dataloader = torch.utils.data.DataLoader(
        test_dataset, batch_size=batch_size, shuffle=False, num_workers=6
    )

    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()

            all_test_outputs.append(outputs.cpu().numpy())
            all_test_labels.append(labels.cpu().numpy())

        average_batch_test_loss = test_loss / len(test_dataloader)
        all_test_outputs = np.concatenate(all_test_outputs)
        all_test_labels = np.concatenate(all_test_labels)

        test_mae = np.mean(np.abs(all_test_outputs - all_test_labels))
        time_str = time.asctime(time.localtime(time.time()))

        print(f'{time_str}, Test MAE: {test_mae:.4f}, Average batch Test MAE: {average_batch_test_loss:.4f}')

    return test_mae, average_batch_test_loss
```

In [57]:
```
def test_r2(model, test_dataset, batch_size, device, criterion):
    model.eval()
    test_loss = 0
    all_test_outputs = []
    all_test_labels = []
    test_dataloader = torch.utils.data.DataLoader(
        test_dataset, batch_size=batch_size, shuffle=False, num_workers=6
    )

    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device).float().view(-1, 1)

            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()

            all_test_outputs.append(outputs.cpu().numpy())
            all_test_labels.append(labels.cpu().numpy())

        average_batch_test_loss = test_loss / len(test_dataloader)
        all_test_outputs = np.concatenate(all_test_outputs).flatten()
        all_test_labels = np.concatenate(all_test_labels).flatten()

        # Calculate R² for the entire test set
        ss_res = np.sum((all_test_labels - all_test_outputs) ** 2)
        ss_tot = np.sum((all_test_labels - np.mean(all_test_labels)) ** 2)
        r2_score = 1 - (ss_res / ss_tot)

        time_str = time.asctime(time.localtime(time.time()))
        print(f'{time_str}, R² Score: {r2_score:.4f}, Average batch Test loss: {average_batch_test_loss:.4f}')

    return r2_score, average_batch_test_loss
```

In [42]:
```
efficientModel = EfficientNet()
file_path = 'efficientModel.pth'

if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    efficientModel.load_state_dict(checkpoint['model_state_dict'])
else:
    efficientModel = training_model(efficientModel, device, 'AdamW', 0.0001, 30, train_dataset, val_dataset, tes
    torch.save({'model_state_dict': efficientModel.state_dict()}, file_path)

efficientModel.to(device)
criterion = torch.nn.MSELoss()
```

```
test_mse, avg_batch_test_loss = test(efficientModel, test_dataset, 128, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\2258279013.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:32:43 2024, Test MSE: 133.6253, Average batch Test loss: 134.0081
```

In [53]:
```
criterion = torch.nn.L1Loss()
efficientModel.to(device)  # Ensure the model is on the correct device

test_mae_value, avg_batch_test_loss = test_mae(
    efficientModel, test_dataset, batch_size=128, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:45:57 2024, Test MAE: 8.4333, Average batch Test MAE: 8.4439
```

In [58]:
```
criterion = torch.nn.MSELoss()

# Ensure the model is on the correct device
efficientModel.to(device)

# Call the test_r2 function
r2_value, avg_batch_test_loss = test_r2(
    efficientModel, test_dataset, batch_size=128, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:49:55 2024, R² Score: 0.4887, Average batch Test loss: 134.0081
```

In [43]:
```
resnetModel = ResNet18()
file_path = 'resnetModel.pth'

if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    resnetModel.load_state_dict(checkpoint['model_state_dict'])
else:
    resnetModel = training_model(resnetModel, device, 'Adam', 0.001, 30, train_dataset, val_dataset, test_datas
    torch.save({'model_state_dict': resnetModel.state_dict()}, file_path)

resnetModel.to(device)
criterion = torch.nn.MSELoss()
test_mse, avg_batch_test_loss = test(resnetModel, test_dataset, 512, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\3208935153.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:37:05 2024, Test MSE: 146.4857, Average batch Test loss: 146.7231
```

In [54]:
```
criterion = torch.nn.L1Loss()
resnetModel.to(device)  # Ensure the model is on the correct device

test_mae_value, avg_batch_test_loss = test_mae(
    resnetModel, test_dataset, batch_size=512, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:47:05 2024, Test MAE: 9.2433, Average batch Test MAE: 9.2505
```

In [59]:
```
criterion = torch.nn.MSELoss()

# Ensure the model is on the correct device
resnetModel.to(device)

# Call the test_r2 function
r2_value, avg_batch_test_loss = test_r2(
    resnetModel, test_dataset, batch_size=512, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:50:18 2024, R² Score: 0.4395, Average batch Test loss: 146.7231
```

In [44]:
```
swinModel = SwinTransformerWithMLP()
file_path = 'swinModel.pth'
```

```
if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    swinModel.load_state_dict(checkpoint['model_state_dict'])
#else:
#    swinModel = training_model(swinModel, device, 'AdamW', 0.001, 30, train_dataset, val_dataset, test_dataset,
#    torch.save({'model_state_dict': swinModel.state_dict()}, file_path)

swinModel.to(device)
criterion = torch.nn.MSELoss()
test_mse, avg_batch_test_loss = test(swinModel, test_dataset, 512, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\3131243850.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:39:05 2024, Test MSE: 178.8896, Average batch Test loss: 178.9268
```

In [55]:
```
criterion = torch.nn.L1Loss()
swinModel.to(device)  # Ensure the model is on the correct device

test_mae_value, avg_batch_test_loss = test_mae(
    swinModel, test_dataset, batch_size=64, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:47:56 2024, Test MAE: 9.9708, Average batch Test MAE: 9.9733
```

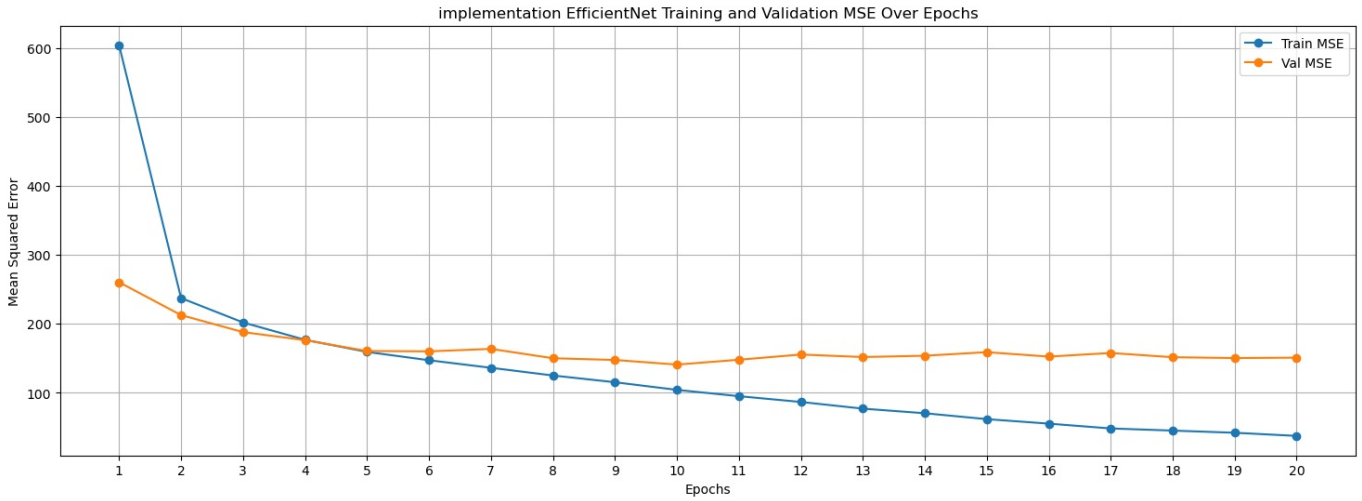
In [60]:
```
criterion = torch.nn.MSELoss()

# Ensure the model is on the correct device
swinModel.to(device)

# Call the test_r2 function
r2_value, avg_batch_test_loss = test_r2(
    swinModel, test_dataset, batch_size=512, device=device, criterion=criterion
)
```

```
Mon Oct 14 23:52:01 2024, R² Score: 0.3155, Average batch Test loss: 178.9268
```

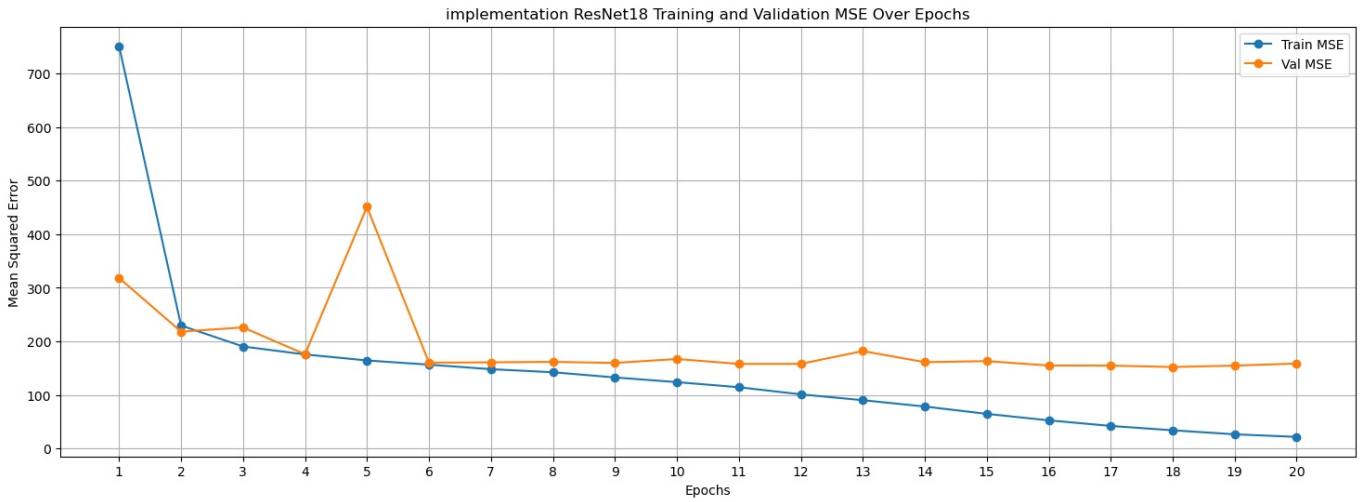
Implentation Evaluation

In [98]:
```
def display_implementation_results(input):
    with open(input, 'r') as file:
        lines = file.readlines()

    data = {
        'epoch': [],
        'train_mse': [],
        'val_mse': []
    }

    for line in lines:
        match = re.search(r'Epoch \[(\d+)/\d+\], Train MSE: ([\d.]+), .* Val MSE: ([\d.]+)', line)
        if match:
            epoch = int(match.group(1))
            train_mse = float(match.group(2))
            val_mse = float(match.group(3))

            data['epoch'].append(epoch)
            data['train_mse'].append(train_mse)
            data['val_mse'].append(val_mse)

    df = pd.DataFrame(data)
    plt.figure(figsize=(18, 6))

    plt.plot(df['epoch'], df['train_mse'], label='Train MSE', marker='o')
    plt.plot(df['epoch'], df['val_mse'], label='Val MSE', marker='o')

    model_name = input.split('.')[0]
    plt.title(model_name+' Training and Validation MSE Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Mean Squared Error')
    plt.xticks(df['epoch'])
    plt.legend()
    plt.grid()

    plt.show()
```

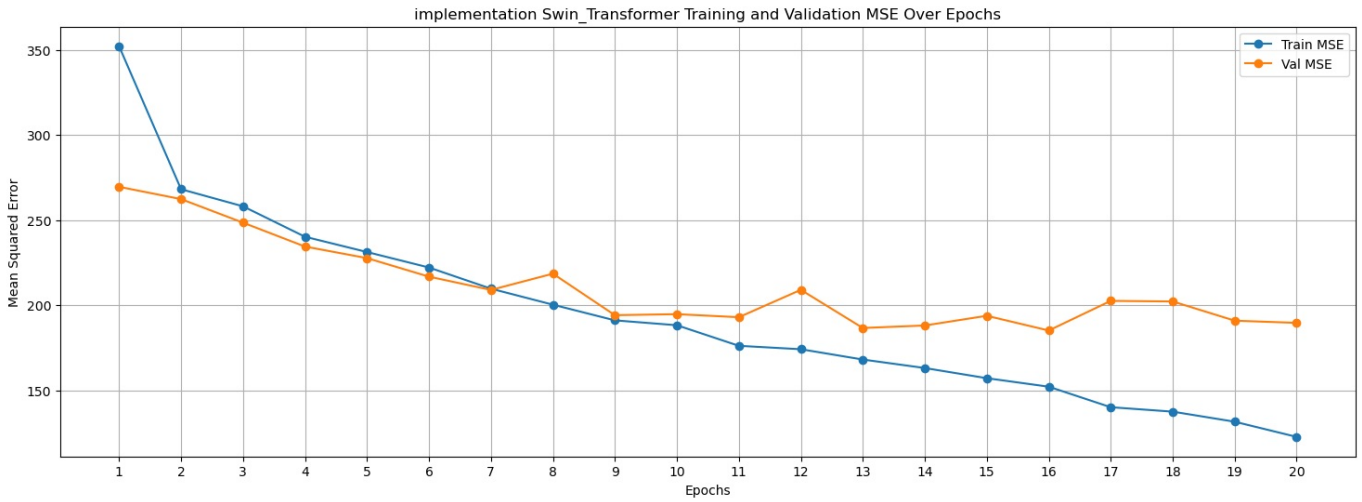
In [99]:
```
display_implementation_results('implementation EfficientNet.txt')
```



In [100…
```
display_implementation_results('implementation ResNet18.txt')
```



In [101…
```
display_implementation_results('implementation Swin_Transformer.txt')
```

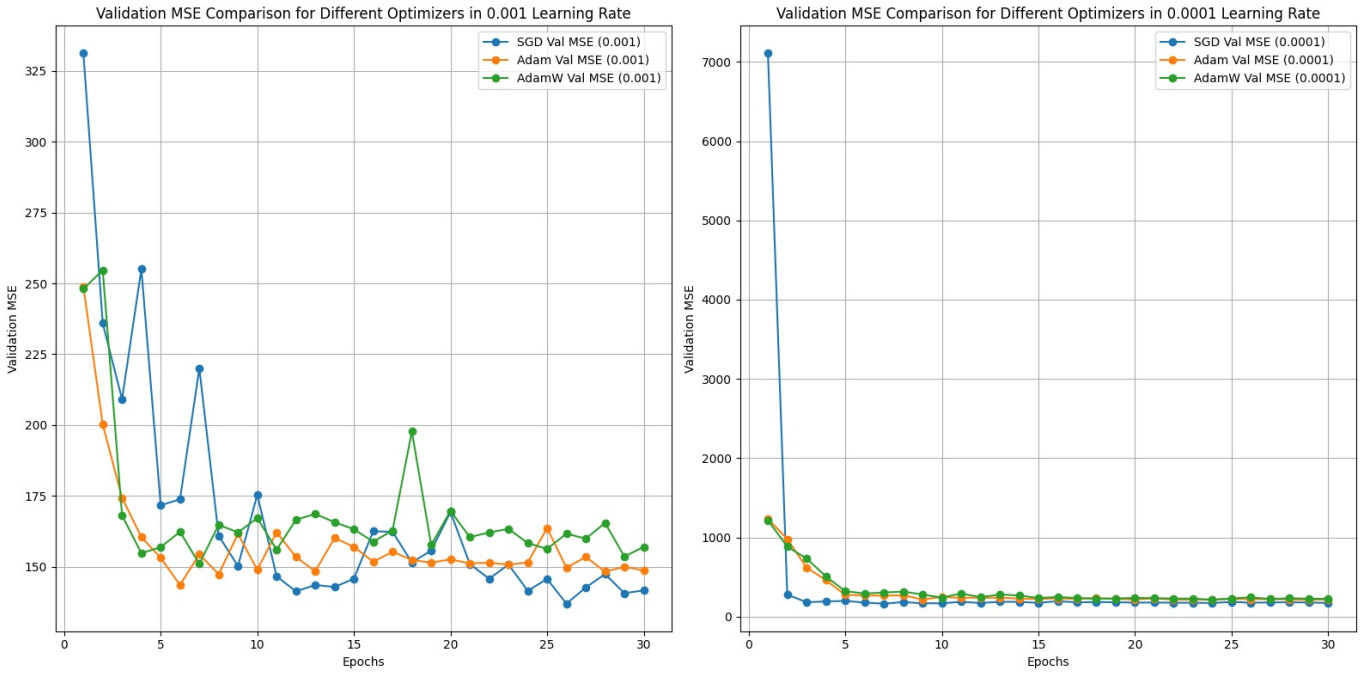


Hyperparameter fine-tuning evaluation

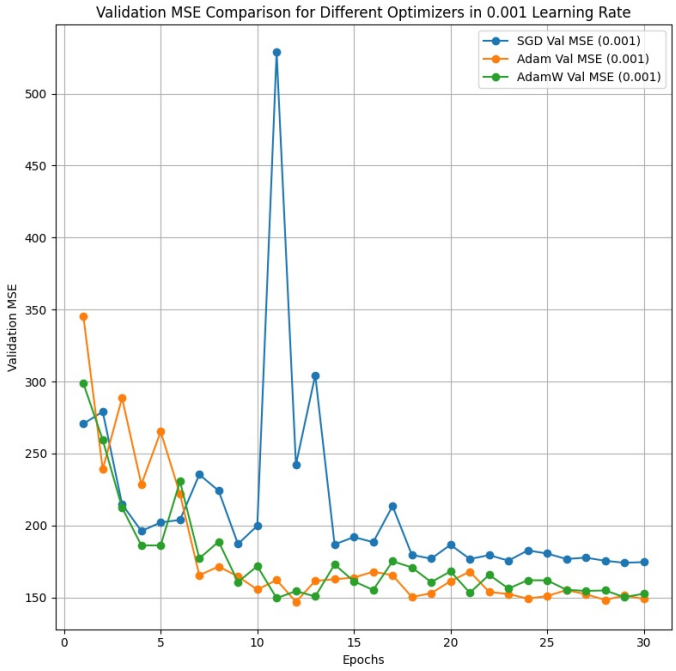
In [3]:
```
def read_log_file(file_path, lr):
    sgd_epoch     = []
    sgd_val_mse   = []

    adam_epoch    = []
    adam_val_mse  = []

    adamw_epoch   = []
    adamw_val_mse = []

    with open(file_path, 'r') as f:
        lines = f.readlines()
```

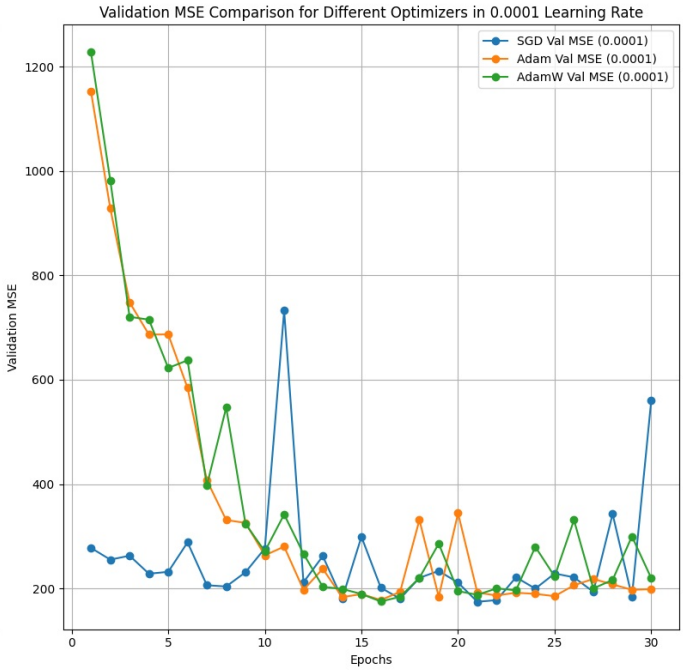
```
        for line in lines:
            match = re.search(r'\{.*?\}, Epoch \[(\d+)/\d+\], Train MSE: [\d.]+, Average batch train mse: [\d.]+
            if match:
                epoch = int(match.group(1))
                val_mse = float(match.group(2))

                if str(lr) in line:
                    if 'SGD' in line:
                        sgd_epoch.append(epoch)
                        sgd_val_mse.append(val_mse)
                    elif 'AdamW' in line:
                        adamw_epoch.append(epoch)
                        adamw_val_mse.append(val_mse)
                    elif 'Adam' in line:
                        adam_epoch.append(epoch)
                        adam_val_mse.append(val_mse)

    return sgd_val_mse, sgd_epoch, adam_val_mse, adam_epoch, adamw_val_mse, adamw_epoch

def display_fine_tuning_evaluation(file_path):
    sgd_val_001, sgd_epoch_001, adam_val_001, adam_epoch_001, adamw_val_001, adamw_epoch_001 = read_log_file(fi
    sgd_val_0001, sgd_epoch_0001, adam_val_0001, adam_epoch_0001, adamw_val_0001, adamw_epoch_0001 = read_log_f

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

    ax1.plot(sgd_epoch_001, sgd_val_001, label='SGD Val MSE (0.001)', marker='o')
    ax1.plot(adam_epoch_001, adam_val_001, label='Adam Val MSE (0.001)', marker='o')
    ax1.plot(adamw_epoch_001, adamw_val_001, label='AdamW Val MSE (0.001)', marker='o')
    ax1.set_title('Validation MSE Comparison for Different Optimizers in 0.001 Learning Rate')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Validation MSE')
    ax1.legend()
    ax1.grid()

    ax2.plot(sgd_epoch_0001, sgd_val_0001, label='SGD Val MSE (0.0001)', marker='o')
    ax2.plot(adam_epoch_0001, adam_val_0001, label='Adam Val MSE (0.0001)', marker='o')
    ax2.plot(adamw_epoch_0001, adamw_val_0001, label='AdamW Val MSE (0.0001)', marker='o')
    ax2.set_title('Validation MSE Comparison for Different Optimizers in 0.0001 Learning Rate')
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('Validation MSE')
    ax2.legend()
    ax2.grid()

    plt.tight_layout()
    plt.show()
```

In [4]:
```
display_fine_tuning_evaluation('efficientModel_results.txt')
```

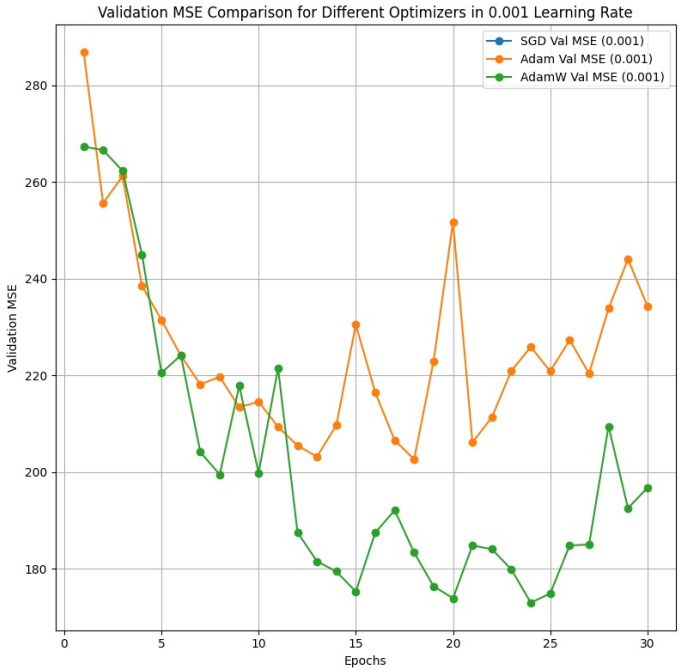


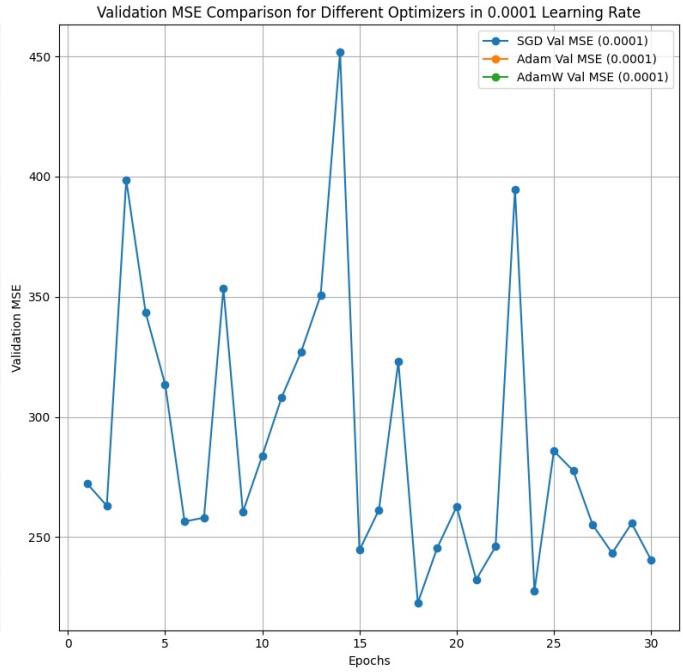
In [5]:
```
display_fine_tuning_evaluation('resnetModel_results.txt')
```





In [6]:
```
display_fine_tuning_evaluation('swinModel_results.txt')
```

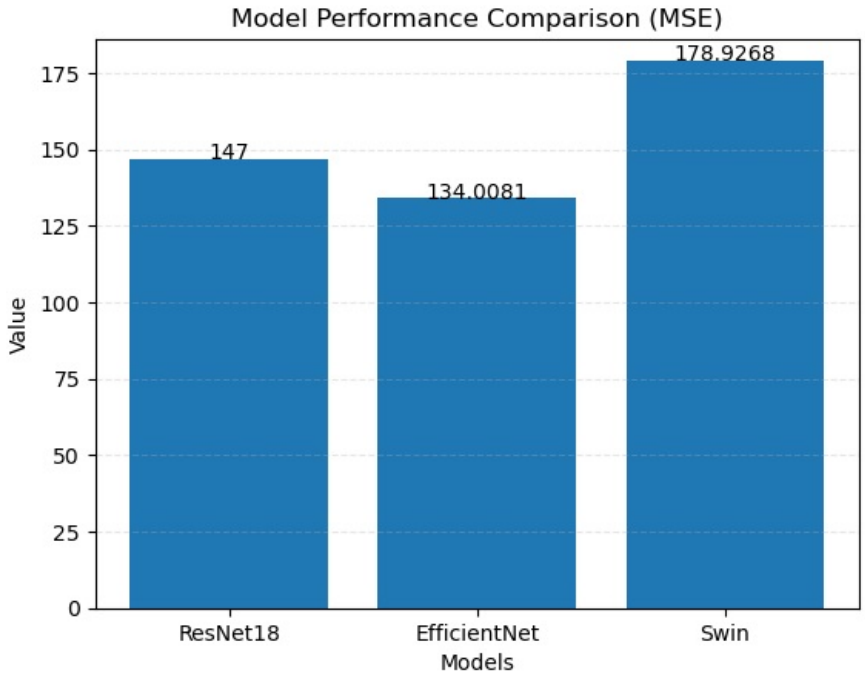




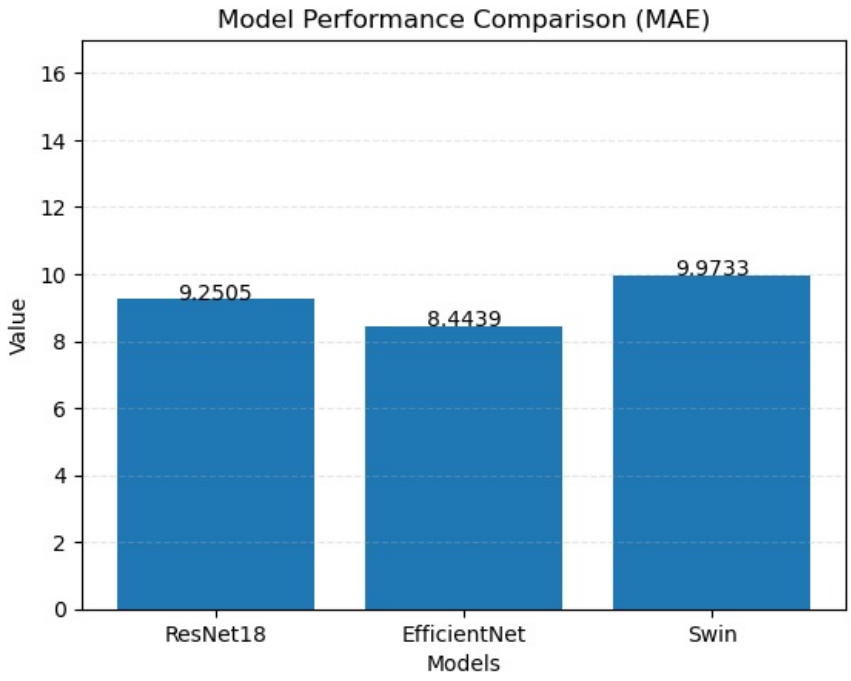
Comparison

In [49]:
```
resNet18_test_mse = 147
efficientNet_test_mse = 134.0081
swin_test_mse = 178.9268
```

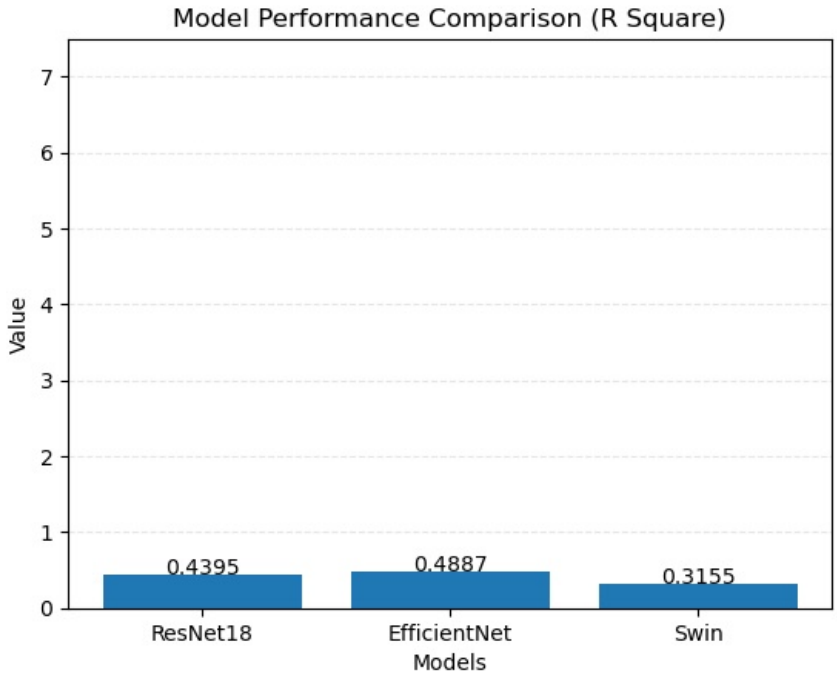
In [51]:
```
def display_evaluation(resNet18, efficientNet, swin, title):
    models = ['ResNet18', 'EfficientNet', 'Swin']
    values = [resNet18, efficientNet, swin]

    plt.bar(np.arange(len(models)), values)

    plt.title(title)
    plt.xlabel('Models')
    plt.ylabel('Value')
    plt.xticks(np.arange(len(models)), models)

    for i, value in enumerate(values):
        plt.text(i, value, str(value), ha='center')

    plt.ylim(0, max(values) + 7)
    plt.grid(axis='y', linestyle='--', alpha=0.3)
    plt.show()

display_evaluation(resNet18_test_mse, efficientNet_test_mse, swin_test_mse, 'Model Performance Comparison (MSE)')
```



In [56]:
```
resNet18_test_mae = 9.2505
efficientNet_test_mae = 8.4439
swin_test_mae = 9.9733
display_evaluation(resNet18_test_mae, efficientNet_test_mae, swin_test_mae, 'Model Performance Comparison (MAE)
```



In [61]:
```
resNet18_test_R_square = 0.4395
efficientNet_test_R_square = 0.4887
swin_test_R_square = 0.3155
display_evaluation(resNet18_test_R_square, efficientNet_test_R_square, swin_test_R_square, 'Model Performance C
```



6. Final models

In [62]:
```
efficientModel = EfficientNet()
file_path = 'efficientModel.pth'

if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    efficientModel.load_state_dict(checkpoint['model_state_dict'])
else:
    efficientModel = training_model(efficientModel, device, 'AdamW', 0.0001, 30, train_dataset, val_dataset, tes
    torch.save({'model_state_dict': efficientModel.state_dict()}, file_path)

efficientModel.to(device)
criterion = torch.nn.MSELoss()
test_mse, avg_batch_test_loss = test(efficientModel, test_dataset, 128, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\1111358782.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:53:20 2024, Test MSE: 133.6253, Average batch Test loss: 134.0081
```

In [63]:
```
resnetModel = ResNet18()
file_path = 'resnetModel.pth'

if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    resnetModel.load_state_dict(checkpoint['model_state_dict'])
else:
    resnetModel = training_model(resnetModel, device, 'Adam', 0.001, 30, train_dataset, val_dataset, test_datas
    torch.save({'model_state_dict': resnetModel.state_dict()}, file_path)

resnetModel.to(device)
criterion = torch.nn.MSELoss()
test_mse, avg_batch_test_loss = test(resnetModel, test_dataset, 512, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\3208935153.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:53:50 2024, Test MSE: 146.4857, Average batch Test loss: 146.7231
```

In [64]:

```
swinModel = SwinTransformerWithMLP()
file_path = 'swinModel.pth'

if os.path.exists(file_path):
    checkpoint = torch.load(file_path)
    swinModel.load_state_dict(checkpoint['model_state_dict'])
#else:
#    swinModel = training_model(swinModel, device, 'AdamW', 0.001, 30, train_dataset, val_dataset, test_dataset,
#    torch.save({'model_state_dict': swinModel.state_dict()}, file_path)

swinModel.to(device)
criterion = torch.nn.MSELoss()
test_mse, avg_batch_test_loss = test(swinModel, test_dataset, 512, device, criterion)
```

```
C:\Users\nizeyu\AppData\Local\Temp\ipykernel_39276\3131243850.py:5: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is poss
ible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default v
alue for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowli
sted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True`
for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any is
sues related to this experimental feature.
  checkpoint = torch.load(file_path)
Mon Oct 14 23:54:12 2024, Test MSE: 178.8896, Average batch Test loss: 178.9268
```