Week 6: Final Project

(Introduction to Deep Learning)

🏫 **University of Colorado, Boulder - Introduction to Deep Learning**

Table of contents

---

1. Task description

Instructions

For the final project, you will identify a Deep Learning problem to perform EDA and model analysis. The project has 140 total points. In the instructions is a summary of the criteria you will use to guide your submission and review others' submissions. You will submit three deliverables:

You will submit three deliverables:

1. **Deliverable 1** — A Jupyter notebook with a description of the problem/data, exploratory data analysis (EDA) procedure, analysis (model building and training), result, and discussion/conclusion. \nSuppose your work becomes so large that it doesn't fit into one notebook (or you think it will be less readable by having one large notebook). In that case, you can make several notebooks or scripts in a GitHub repository (as deliverable 3) and submit a report-style notebook or pdf instead. If your project doesn't fit into Jupyter notebook format (E.g., you built an app that uses ML), write your approach as a report and submit it in a pdf form.

2. **Deliverable 2** — A public project GitHub repository with your work (please also include the GitHub repo URL in your notebook/report).

3. **Deliverable 3** — A video presentation or demo of your work. The minimum video length is 5 min, the maximum length is 15 min. The recommended length is about 10 min. Submit the video in .mp4 format. The

presentation should be a condensed version as if you're doing a short pitch to advertise your work, so please focus on the highlights:

```
* What problem do you solve?

* What ML approach do you use, or what methods does your app use?

* Show the result or run an app demo.
```

Data byproduct

If your project creates data and you want to share it, an excellent way to share would be through a Kaggle dataset or similar. Similarly, suppose you want to make your video public. In that case, we recommend uploading it to YouTube or similar and posting the link(s) to your repository or blog instead of a direct upload to GitHub. It is generally a good practice not to upload big files to a Git repository.

Grading review overview

Three of your peers will review each of your three deliverables (Jupyter notebook or pdf report, video presentation, and GitHub repository) based on the rubrics for each deliverable.

Use the rubrics to guide your project to include all parts for the grade you want to achieve. The project has 140 total points.

Instructions: Step 1

Gather data, determine the method of data collection and provenance of the data (1 point)

In the earliest phase, select a data source and problem. Feel free to share and discuss your idea on the class discussion board.

Instructions: Step 2

Identify a Deep Learning Problem (5 points)

If you're going to use a Kaggle competition or similar, you must focus more on model building and/or analysis to be a valid project. Replicating what's in the Kaggle kernel or other notebooks available online is not a valid project. It is reasonable to add different approaches and compare them with the Kaggle kernel or other notebooks available online. It is also good to find a research paper, implement an algorithm, and run experiments comparing its performance to different algorithms.

Instructions: Step 3

Exploratory Data Analysis (EDA) - Inspect, Visualize, and Clean the Data (34 points)

Go through the initial data cleaning and EDA and judge whether you need to collect more or different data.

EDA Procedure Example:

- Describe the factors or components that make up the dataset (The "factors" here are called "features" in the machine learning term. These factors are often columns in the tabulated data). For each factor, use a box-plot, scatter plot, histogram, etc., to describe the data distribution as appropriate.
- Describe correlations between different factors of the dataset and justify your assumption that they are correlated or not correlated. You may use numeric or qualitative/graphical analysis for this step.
- Determine if any data needs to be transformed. For example, if you're planning on using an SVM method for prediction, you may need to normalize or scale the data if there is a considerable difference in the range of the data.
- Using your hypothesis, indicate if it's likely that you should transform data, such as using a log transform or other transformation of the dataset.

- You should determine if your data has outliers or needs to be cleaned in any way. Are there missing data values for specific factors? How will you handle the data cleaning? Will you discard, interpolate or otherwise substitute data values?
- If you believe that specific factors will be more important than others in your analysis, you should mention which and why. You will use this to confirm your intuitions in your final write-up.

Instructions: Step 4

Perform Analysis Using Deep Learning Models of your Choice, Present Discussion, and Conclusions (65 points)

Start the main analysis (the main analysis refers to supervised learning tasks such as classification or regression). Depending on your project, you may have one model or more. Generally, it is deemed a higher quality project if you compare multiple models and show your understanding of why specific models work better than the other or what limitations or cautions specific models may have. For machine learning models, another recommendation is to show enough effort on the hyperparameter optimization.

If your project involves making a web app (not required), you can include the demo.

Instructions: Step 5

Produce Deliverables: High-Quality, Organized Jupyter Notebook Report, Video presentation, and GitHub Repository (35 points)

These deliverables serve two purposes- grade for this course and your project portfolio that you can show when you apply for jobs.

If you haven't used GitHub previously, please find a tutorial and get acquainted with it before the project deadline. For the sake of this project, you can use GitHub to showcase your codebase. In the real world, versioning with GitHub is vital for collaboration. Sometimes Jupyter notebooks don't seem particularly well-suited to versioning with GitHub due to hard-to-read diffs and the like. If you want to use this project as an opportunity to practice versioning with GitHub, consider something like the following: https://www.reviewnb.com.

One of the essential components of this project is peer review. As a professional data scientist, a critical part of your job will likely involve communicating results to key stakeholders and convincing decision-makers of your conclusions. To further your professional development, think of your peers as work colleagues and utilize your report, video presentation, and GitHub repository to communicate with them. Imagine that you are writing code that co-workers will collaborate on and maintain, so make sure to organize and appropriately comment on the codebase. Reviewing your peers' work also has critical professional value. In your career, you will maintain or work with existing code. As you assess your peers' projects, imagine that you will be collaborating with them and need to understand their codebase and evaluate whether their results lead to the conclusions they claim.

Project's Description

Facial Keypoints Detection Kaggle Dataset

https://www.kaggle.com/c/facial-keypoints-detection/overview

Description

The objective of this task is to predict keypoint positions on face images. This can be used as a building block in several applications, such as:

- Tracking faces in images and video
- Analysing facial expressions
- Detecting dysmorphic facial signs for medical diagnosis
- Biometrics / face recognition

Detecing facial keypoints is a very challenging problem. Facial features vary greatly from one individual to another, and even for a single individual, there is a large amount of variation due to 3D pose, size, position, viewing angle, and illumination conditions. Computer vision research has come a long way in addressing these difficulties, but there remain many opportunities for improvement.

Acknowledgements

The data set for this competition was graciously provided by Dr. Yoshua Bengio of the University of Montreal. The tutorial was developed by James Petterson.

Problem Description

The problem at hand can be articulated as follows: We have a set of images and aim to identify the locations of facial keypoints within these images. The keypoints include the centers, outer and inner corners of the eyes, corners of the mouth along with the top and bottom center of the lips, outer and inner corners of the eyebrows, and finally, the nosetip.

Our objective is to predict the positions of these facial features in the images. To achieve this, we will develop an algorithm that produces (x, y)-value pairs, denoting the pixel-based positions of the respective keypoints.

This task holds significant relevance from various perspectives. Firstly, it serves as an excellent learning opportunity in the realm of automated image processing and convolutional neural networks. Moreover, feature recognition in images is a crucial foundation for numerous machine learning algorithms employed in applications such as production and measurement, as well as in domains like autonomous cars and facial recognition.

---

2. Import and Data Loading

Import Data

The provided dataset was obtained from a Kaggle competition accessible through the following link: https://www.kaggle.com/c/facial-keypoints-detection/overview. The dataset comprises four files, with two of them being zip archives containing the training and test data. The remaining two files consist of a sample submission, outlining the desired format for submitting results to the Kaggle leaderboard, and a lookup table that details the specific positions of facial features in each image. The latter must be submitted as part of the competition requirements.

In the subsequent sections, we will delve into a more comprehensive description of the given data, specifically focusing on the images.

In [1]:
```
# General Data Science Packages

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Utility and Environment Packages

import os
import zipfile

from pathlib import Path

# Machine Learning Packages

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import LearningRateScheduler
```

```
# Plotting

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Setup path to a data folder
data_path = Path("data/")
image_path = data_path / "facial-keypoints-detection"

# If the image folder doesn't exist, download it and prepare it
if image_path.is_dir():
    print(f"{image_path} directory already exists...")
else:
    print(f"{image_path} does not exist, creating one...")
    image_path.mkdir(parents=True, exist_ok=True)
```

```
data\facial-keypoints-detection directory already exists...
```

In [3]:
```
def walk_through_dir(dir_path):
    """ Walks through dir_path returning its contents."""
    for dirpath, dirnames, filenames in os.walk(dir_path):
        print(f"There are {len(dirnames)} directories and {len(filenames)} files in '{dirpath}")
```

In [4]:
```
walk_through_dir(image_path)
```

```
There are 0 directories and 4 files in 'data\facial-keypoints-detection
```

In [5]:
```
# Setup train and testing paths
train_data = image_path  / "training.zip"
test_data = image_path / "test.zip"
```

In [6]:
```
df_train = pd.read_csv(train_data)
df_test = pd.read_csv(test_data)
```

Data Description

In [7]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7049 entries, 0 to 7048
Data columns (total 31 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   left_eye_center_x          7039 non-null   float64
 1   left_eye_center_y          7039 non-null   float64
 2   right_eye_center_x         7036 non-null   float64
 3   right_eye_center_y         7036 non-null   float64
 4   left_eye_inner_corner_x    2271 non-null   float64
 5   left_eye_inner_corner_y    2271 non-null   float64
 6   left_eye_outer_corner_x    2267 non-null   float64
 7   left_eye_outer_corner_y    2267 non-null   float64
 8   right_eye_inner_corner_x   2268 non-null   float64
 9   right_eye_inner_corner_y   2268 non-null   float64
 10  right_eye_outer_corner_x   2268 non-null   float64
 11  right_eye_outer_corner_y   2268 non-null   float64
 12  left_eyebrow_inner_end_x   2270 non-null   float64
 13  left_eyebrow_inner_end_y   2270 non-null   float64
 14  left_eyebrow_outer_end_x   2225 non-null   float64
 15  left_eyebrow_outer_end_y   2225 non-null   float64
 16  right_eyebrow_inner_end_x  2270 non-null   float64
 17  right_eyebrow_inner_end_y  2270 non-null   float64
 18  right_eyebrow_outer_end_x  2236 non-null   float64
 19  right_eyebrow_outer_end_y  2236 non-null   float64
 20  nose_tip_x                 7049 non-null   float64
 21  nose_tip_y                 7049 non-null   float64
 22  mouth_left_corner_x        2269 non-null   float64
 23  mouth_left_corner_y        2269 non-null   float64
 24  mouth_right_corner_x       2270 non-null   float64
 25  mouth_right_corner_y       2270 non-null   float64
 26  mouth_center_top_lip_x     2275 non-null   float64
 27  mouth_center_top_lip_y     2275 non-null   float64
 28  mouth_center_bottom_lip_x  7016 non-null   float64
 29  mouth_center_bottom_lip_y  7016 non-null   float64
 30  Image                      7049 non-null   object
dtypes: float64(30), object(1)
memory usage: 1.7+ MB
```

In [8]:
```
df_train.describe()
```

Out[8]:

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inner_corner_x	left_ey
count	7039.000000	7039.000000	7036.000000	7036.000000	2271.000000	
mean	66.359021	37.651234	30.306102	37.976943	59.159339	
std	3.448233	3.152926	3.083230	3.033621	2.690354	
min	22.763345	1.616512	0.686592	4.091264	19.064954	
25%	65.082895	35.900451	28.783339	36.327681	58.039339	
50%	66.497566	37.528055	30.251378	37.813273	59.304615	
75%	68.024752	39.258449	31.768334	39.566729	60.519810	
max	94.689280	80.502649	85.039381	81.270911	84.440991	

8 rows × 30 columns

In [9]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1783 entries, 0 to 1782
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   ImageId  1783 non-null   int64
 1   Image    1783 non-null   object
dtypes: int64(1), object(1)
memory usage: 28.0+ KB
```

In [10]:
```
def unpack_image(im):
    """
    Converts a string of grey-scale values per pixel into a NumPy array of shape 96x96 with integer va

    Parameters:
    - im (str): A string containing grey-scale values for pixels.

    Returns:
    - list: A list representation of the NumPy array, as Pandas does not support NumPy arrays as value
    """
    im = (im.split())
    im = np.array(im, dtype='int')
    im = np.reshape(im, (96, 96))
    return im.tolist()
```

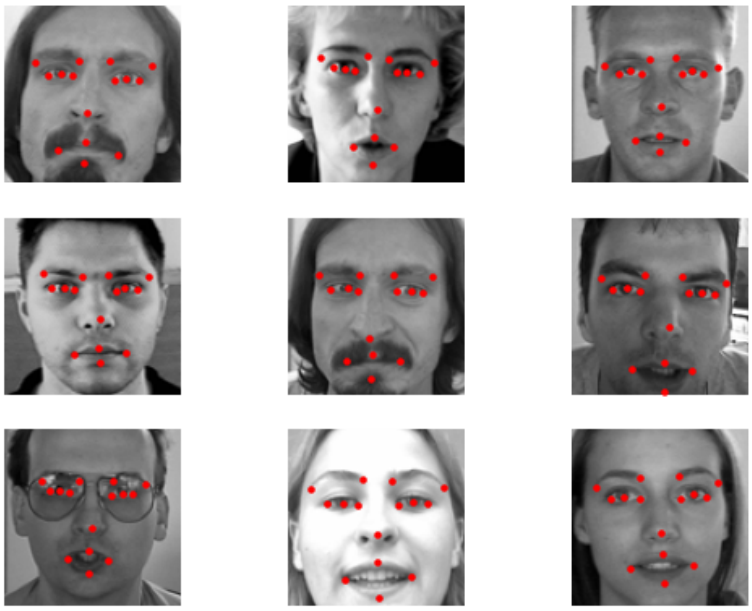
In [11]:
```
def extract_series(df):
    """
    Extracts individual series from a pandas DataFrame, separating them into features and targets.

    Parameters:
    - df (pd.DataFrame): A pandas DataFrame containing both image data and corresponding target values

    Returns:
    - tf.data.Dataset: A TensorFlow dataset created from the extracted features and targets.
    """
    # Extracting image data and targets
    images = df.iloc[:,-1].apply(lambda x: unpack_image(x))
    targets = df.iloc[:, 0:-1]

    # Resolving lists in the global np.array and converting to a np.array
    images = np.array(images.to_numpy().tolist())

    # Creating a TensorFlow dataset from the extracted data
    dataset = tf.data.Dataset.from_tensor_slices((images, targets))

    return dataset
```

In [12]:
```
# Creating a tensor flow dataset from the data
dataset_training = extract_series(df_train)
```

From the dataset, we observe that it consists of 96x96 pixel-sized grayscale images. Each pixel within these images possesses a value ranging from zero to 255, covering the standard grayscale color spectrum. Additionally, the dataset comprises 7,049 training images and 1,783 test images. Each image is associated with 30 target values, representing the x, y-position of the facial keypoints mentioned earlier. These target values are of data type float, signifying the position of the features, and hence, they can assume values in between two pixels.

In the upcoming section, we will visualize some examples from the training dataset to gain a better understanding of the diverse image types present in the dataset and assess the accuracy of the provided target positions.

In [13]:
```
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(list(dataset_training.as_numpy_iterator())[i][0],cmap='gray', vmin=0, vmax=255)
    plt.scatter(list(dataset_training.as_numpy_iterator())[i][1][0::2], list(dataset_training.as_numpy
    plt.axis("off")
```



From this small sample of images, it's evident that there are instances of people wearing glasses. Unfortunately, the dataset lacks metadata to identify which pictures depict individuals with glasses. It would be intriguing to explore how this factor influences the quality of predictions. Additionally, it's worth noting that the dataset encompasses both male and female subjects. Understanding and addressing these factors could contribute to refining the predictive quality of our model.

Data Preparation

As an initial step, we will undertake the task of cleaning our dataset. It has been established that all input arrays are representations of valid 96x96 pixel images. This validity is crucial, as any deviation would have resulted in errors during the dataset creation process, given its requirement for 96x96 input values.

Nevertheless, up to this point, we have not verified the validity and completeness of the provided target positions. This aspect remains an essential consideration in ensuring the overall quality and reliability of our dataset.

In [14]:
```
df_train[0:-1].isna().sum()
```

Out[14]:
```
left_eye_center_x              10
left_eye_center_y              10
right_eye_center_x             13
right_eye_center_y             13
left_eye_inner_corner_x      4777
left_eye_inner_corner_y      4777
left_eye_outer_corner_x      4781
left_eye_outer_corner_y      4781
right_eye_inner_corner_x     4780
right_eye_inner_corner_y     4780
right_eye_outer_corner_x     4780
right_eye_outer_corner_y     4780
left_eyebrow_inner_end_x     4778
left_eyebrow_inner_end_y     4778
left_eyebrow_outer_end_x     4823
left_eyebrow_outer_end_y     4823
right_eyebrow_inner_end_x    4778
right_eyebrow_inner_end_y    4778
right_eyebrow_outer_end_x    4812
right_eyebrow_outer_end_y    4812
nose_tip_x                      0
nose_tip_y                      0
mouth_left_corner_x          4779
mouth_left_corner_y          4779
mouth_right_corner_x         4778
mouth_right_corner_y         4778
mouth_center_top_lip_x       4773
mouth_center_top_lip_y       4773
mouth_center_bottom_lip_x      33
mouth_center_bottom_lip_y      33
Image                           0
dtype: int64
```

In [15]:
```
df_train_clean = df_train.dropna()
df_train_clean.shape
```

Out[15]:
```
(2140, 31)
```

In [16]:
```
# Creating a tensor flow dataset from the data
df_training_clean = extract_series(df_train_clean)
```

---

3. Exploratory Data Analysis (EDA)

Position Histograms

During the Exploratory Data Analysis (EDA) procedure, I will address several questions to gain insights into the characteristics and variability of the given input data. The questions are outlined as follows:

---

1. **What is the distribution of the x and y position of the nose tip on the images?**
   - This analysis aims to provide information about the centering of the images.

---

2. **What is the distribution of the outer and upper/lower most key features on the images?**
   - This examination seeks to offer insights into the distribution of the size of the subjects on the images.

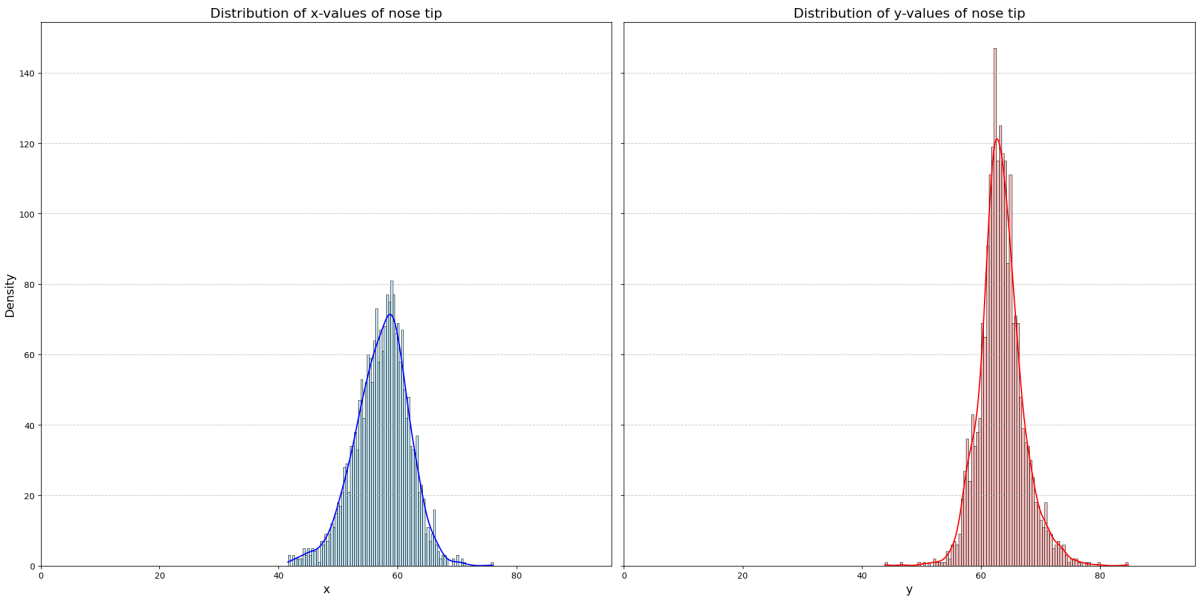
---

3. **What is the distribution of the mean gray-scale value for the images?**
   - This investigation is designed to reveal information about the variability in lighting conditions across the images.

Addressing these questions will contribute to a better understanding of the spatial characteristics, subject sizes, and lighting conditions present in the dataset.

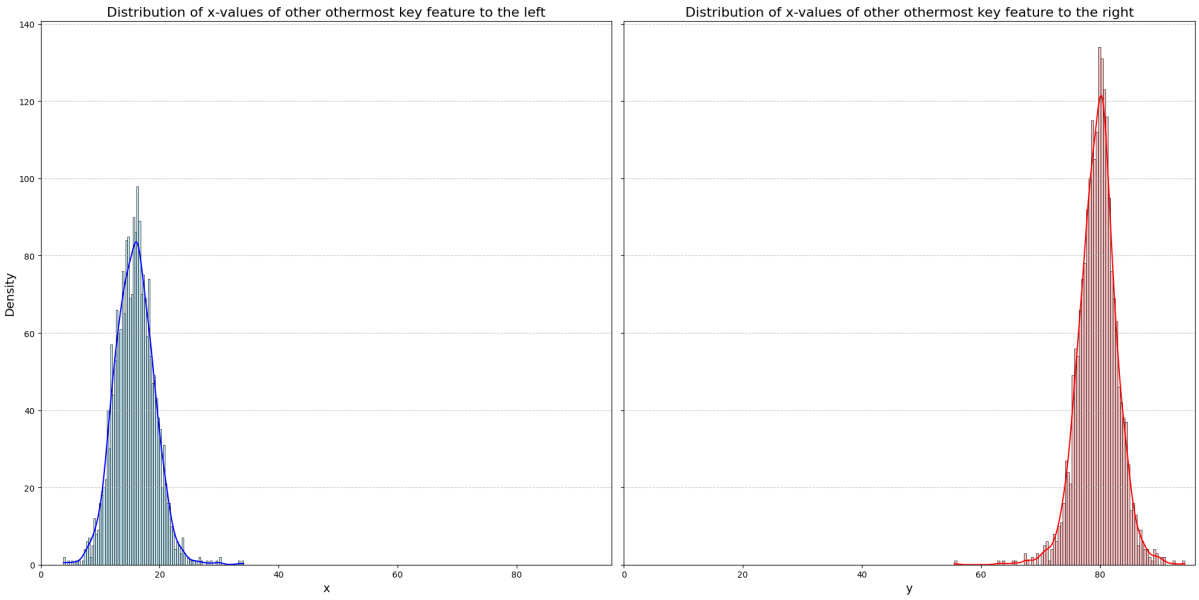
In [17]:
```
def plot_feature(features, titles):
    """
    Plots the distribution of x and y values for a specific facial feature.

    Parameters:
    - features (list): A list containing two arrays - x and y values of a facial feature.
    - titles (list): A list containing two strings - titles for the x and y value plots.

    Returns:
    None
    """
    # Creating a single figure with two subplots side by side
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10), sharey=True)

    # Plotting the distribution of x-values of the facial feature
    sns.histplot(features[0], kde=True, bins=96, ax=ax1, color='skyblue')
    ax1.lines[0].set_color('blue')
    ax1.set_title(titles[0], fontsize=16)
    ax1.set_xlabel('x', fontsize=14)
    ax1.set_ylabel('Density', fontsize=14)
    ax1.set_xlim(0, 96)
    ax1.grid(axis='y', linestyle='--', alpha=0.7)

    # Plotting the distribution of y-values of the facial feature
    sns.histplot(features[1], kde=True, bins=96, ax=ax2, color='lightcoral', edgecolor='black')
    ax2.lines[0].set_color('red')
    ax2.set_title(titles[1], fontsize=16)
    ax2.set_xlabel('y', fontsize=14)
    ax2.set_ylabel('Density', fontsize=14)
    ax2.set_xlim(0, 96)
    ax2.grid(axis='y', linestyle='--', alpha=0.7)

    # Adjusting layout for better spacing
    plt.tight_layout()

    # Displaying the figure
    plt.show()
```

In [18]:
```
# Extracting nose x and y coordinates from the training dataset
nose_x = [x[1][21] for x in list(df_training_clean.as_numpy_iterator())]
nose_y = [x[1][22] for x in list(df_training_clean.as_numpy_iterator())]

plot_feature((nose_x, nose_y), ('Distribution of x-values of nose tip', 'Distribution of y-values of
```

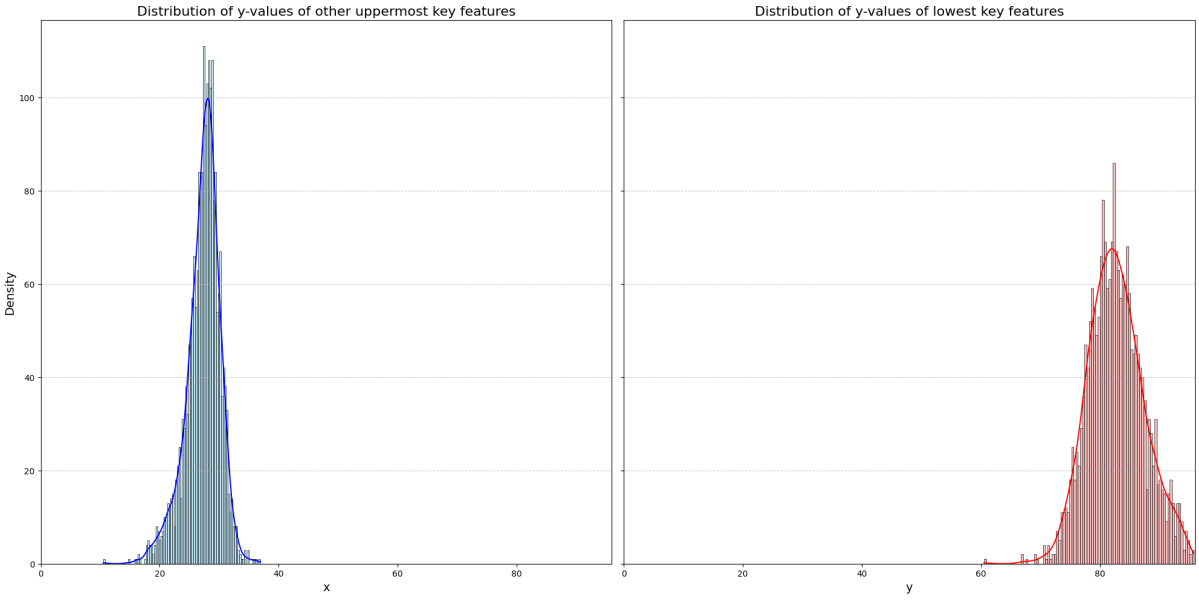


From the histograms, it is apparent that the distributions for the x and y positions of the nose tip are relatively narrow. Notably, the histogram for the x coordinates appears to be wider, indicating a higher degree of variability in the training dataset concerning the centering of faces in the x-direction compared to the y-direction. This insight suggests that there may be more variation in the horizontal positioning of facial features, highlighting the importance of considering and addressing such nuances during the model training process.

In [19]:
```
outer_left = [min(x[1][0::2]) for x in list(df_training_clean.as_numpy_iterator())]
outer_right = [max(x[1][0::2]) for x in list(df_training_clean.as_numpy_iterator())]

features = (outer_left, outer_right)
titles = ('Distribution of x-values of other othermost key feature to the left', 'Distribution of x-va

plot_feature(features, titles)
```

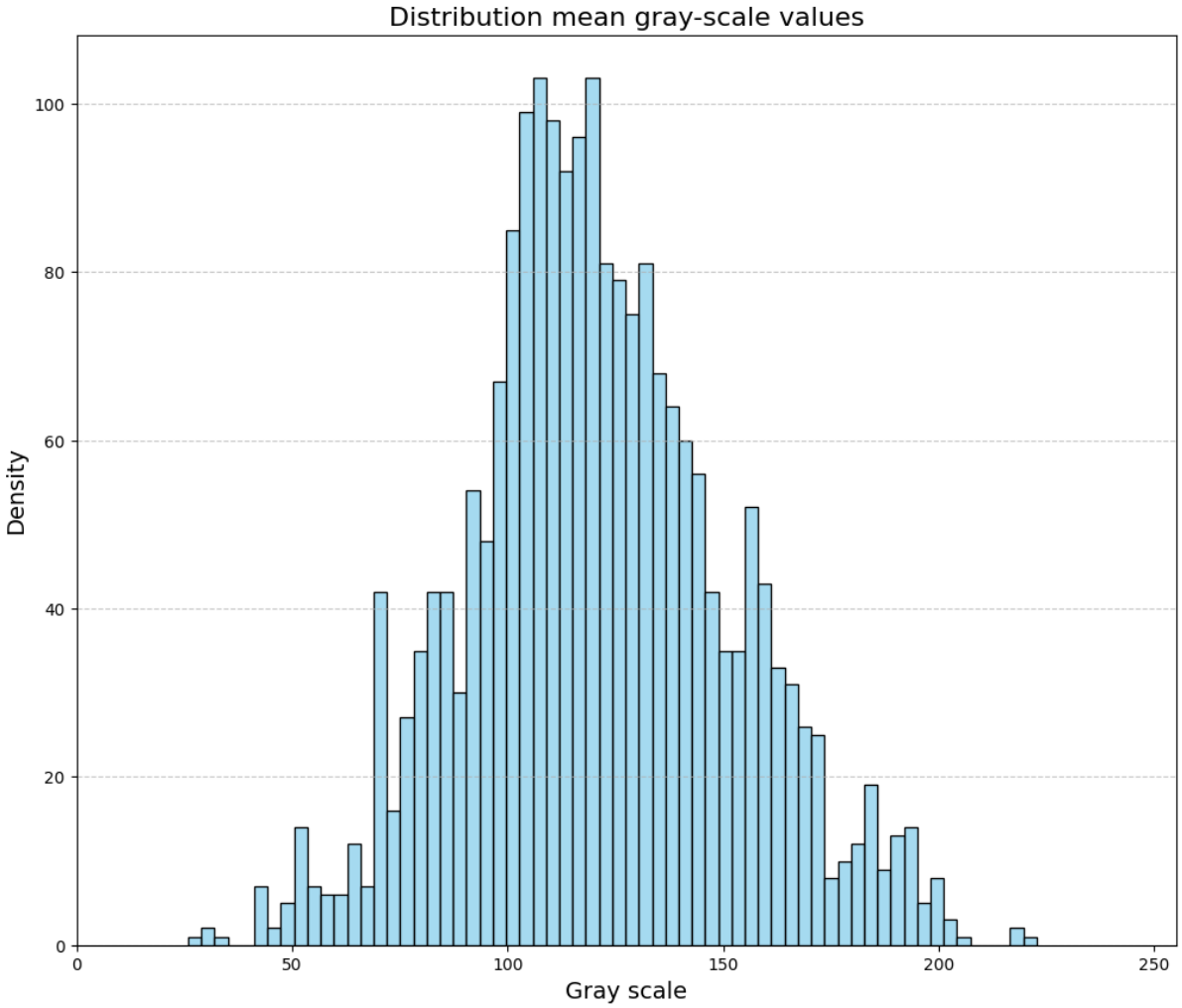


In [20]:
```
lower = [max(x[1][1::2]) for x in list(df_training_clean.as_numpy_iterator())]
upper = [min(x[1][1::2]) for x in list(df_training_clean.as_numpy_iterator())]

features = (upper, lower)
titles = ('Distribution of y-values of other uppermost key features', 'Distribution of y-values of lou

plot_feature(features, titles)
```



The presented histograms indicate that the key features are generally depicted in nearly full size on most images. Even in the extreme values of the respective histograms, the key features tend to lie near the boundaries of the images. An exception is observed for the uppermost key features, which often appear somewhat distant from the image boundary. However, this is reasonable, as the uppermost key feature likely represents a portion of the eyebrows. In the vertical dimension, faces do not conclude with the eyebrows; they are followed by the hair. Conversely, the lowermost key feature, representing some part of the lip, is not followed by any features below besides the chin. This observation implies that the faces in the training images are generally centered, with the eyebrows positioned towards the upper part of the images and the chin towards the lower part.

In [21]:
```
mean_gray = [np.mean(x[0]) for x in list(df_training_clean.as_numpy_iterator())]

# Plotting the distribution of x-values of the facial feature
plt.figure(figsize=(12, 10))
sns.histplot(mean_gray, bins=64, color='skyblue')
plt.title('Distribution mean gray-scale values', fontsize=16)
plt.xlabel('Gray scale', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.xlim(0,255)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```



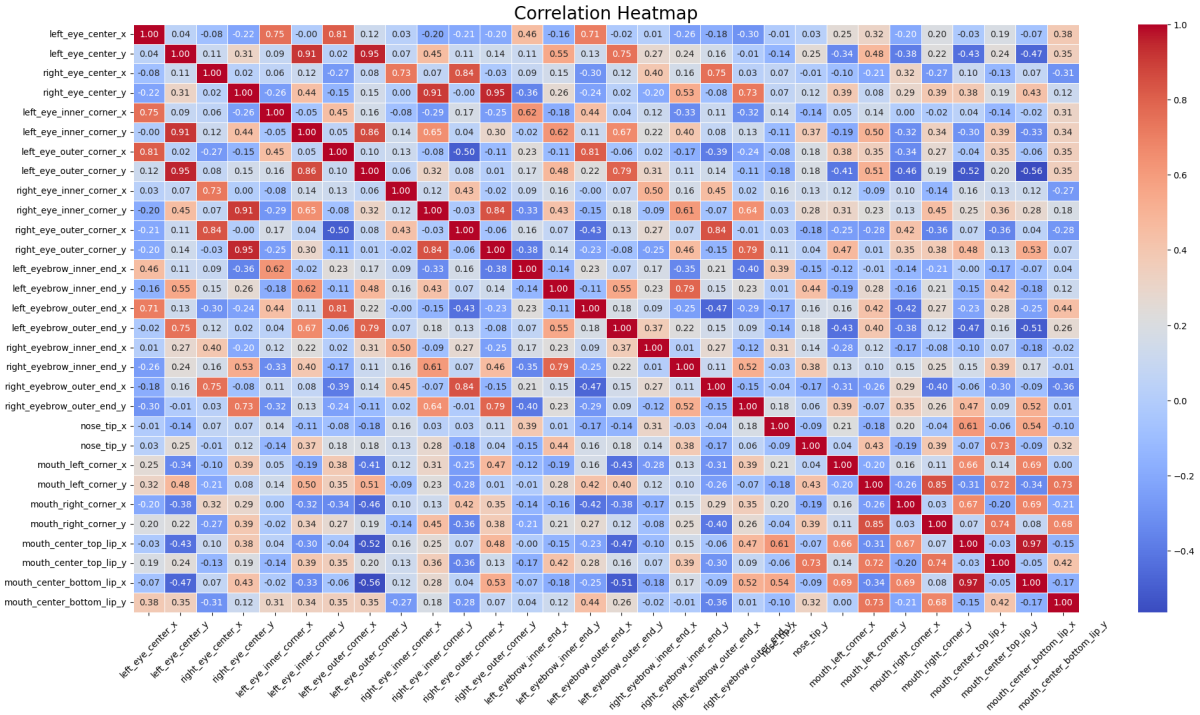
The observation that the mean gray-scale value falls near the center of the 0-255 range suggests that the majority of training images are well-exposed, with some images being slightly darker or brighter.

As a concluding step in our Exploratory Data Analysis (EDA) procedure, we will generate a correlation heatmap between our target variables, representing the positions of the key features. This heatmap will provide insights into the interdependencies among the facial keypoints, aiding in our understanding of how changes in one key feature might correlate with changes in others.

Descriptive Statistics

In [22]:
```
cor_mat = df_train_clean.iloc[:,0:-1].corr()

# Set up the figure with a larger size
plt.figure(figsize=(24, 12))

sns.heatmap(cor_mat, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

# Set the title and adjust font size
plt.title('Correlation Heatmap', fontsize=20)
```

```
# Rotate x-axis labels for better visibility
plt.xticks(rotation=45)

# Show the plot
plt.show()
```



The correlation heatmap reveals that there are naturally strong correlations between the positions of key features on faces. This is to be expected, as, for instance, the center of one eye on standardized images (centered, full-face, well-exposed as demonstrated earlier) should logically exhibit some level of correlation with the corners of the respective eye. The observed correlations reflect the inherent spatial relationships among facial features, providing valuable insights into the coordinated movements and placements of these key points.

---

4. Model Architecture

Model Building

To prepare our input data for the planned Convolutional Neural Network (CNN), we need to adapt its structure. The convolutional layers expect an input data shape of (None, width, height, layer). In our case, since we are dealing with gray-scale images, the last dimension (layer) is of size 1. The first dimension (None) represents the batch size. By restructuring our input data in this way, we align with the requirements of the CNN architecture we plan to use.

In [256…
```
BATCH_SIZE = 16
EPOCHS = 10
```

In [246…
```
# Dataset Creation training data

# Extract single serieses from pd.Dataframe as features and targets
train_images = df_train_clean.iloc[:,-1].apply(lambda x: unpack_image(x))
train_targets = df_train_clean.iloc[:,0:-1]

# Hack to resolve the underlying lists in the global np.array and return
# the whole structure as a np.array
train_images = np.array(train_images.to_numpy().tolist())
train_images = np.expand_dims(train_images, 3) # Add 1 dimension - layer
train_images = np.reshape(train_images, (2140,96,96,1)) # maybe this is not necessary
```

```
# Creating a tensor flow dataset from the data
dataset_training_clean = tf.data.Dataset.from_tensor_slices((train_images,train_targets))

# Dataset Creation test data

# Extract single serieses from pd.Dataframe as features and targets
test_images = df_test.iloc[:,-1].apply(lambda x: unpack_image(x))

# Hack to resolve the underlying lists in the global np.array and return
# the whole structure as a np.array
test_images = np.array(test_images.to_numpy().tolist())
test_images = np.expand_dims(test_images, 3)
test_images = np.reshape(test_images, (len(df_test),96,96,1)) # maybe this is not necessary

# Creating a tensorflow dataset from the data
dataset_testing = tf.data.Dataset.from_tensor_slices(test_images)
```

We'll encapsulate the batch creation process, where datasets are sliced into batches, within a function. This allows us to automate the process and experiment with various batch sizes during hyperparameter optimization.

In [247…

```
def create_batches(BATCH_SIZE, train, dataset_train, dataset_test):
    """
    Creates batches of data for training, validation, and testing.

    Parameters:
    - BATCH_SIZE (int): The desired batch size for training and validation datasets.
    - train (int): The size of the training dataset.
    - dataset_train (tf.data.Dataset): TensorFlow dataset containing training data.
    - dataset_test (tf.data.Dataset): TensorFlow dataset containing test data.

    Returns:
    - tuple: A tuple containing TensorFlow datasets for training, validation, and testing.
      - dataset_train (tf.data.Dataset): Batched training dataset.
      - dataset_val (tf.data.Dataset): Batched validation dataset.
      - dataset_test (tf.data.Dataset): Batched test dataset.
    """
    train_size = int(len(train) * 0.8)
    dataset = dataset_train.take(train_size)
    dataset = dataset.batch(BATCH_SIZE)
    dataset_val = dataset_train.skip(train_size).take(-1)
    dataset_val = dataset_val.batch(BATCH_SIZE)

    dataset_test = dataset_test.batch(1)

    return dataset, dataset_val, dataset_test
```

Now, we will define our initial Convolutional Neural Network (CNN) architecture to address the given problem. The architecture involves three 2D convolutional blocks with increasing layer sizes and a 3x3 filter, along with max-pooling of size 2x2. The output of the convolutional layers will be processed by a rectified linear unit (ReLU) activation function. Following the convolutional layers, the model will include a hidden dense layer and a dense output layer, resulting in a final output size of 30 parameters.

For optimization, we will utilize the Adam algorithm with the default learning rate of 0.001. Given that we are dealing with a regression problem, the mean squared error will serve as the loss function. This choice penalizes larger deviations more, aiding the learning procedure in approximating the target solution. The metric for evaluating prediction quality will be the mean absolute error, providing an intuitively interpretable measure as the mean distance between the target and predicted feature positions.

In [248…

```
def step_decay(epoch, learning_rate):
    """
    Learning rate schedule using step decay.

    This function defines a learning rate schedule that decreases the learning rate
    by a specified factor after a certain number of epochs. The formula used for
    the schedule is: lr = initial_lr * drop^(floor((1 + epoch) / epochs_drop))

    Parameters:
    - epoch (int): The current epoch number.
```

```
    Returns:
    - float: The updated learning rate for the given epoch.
    """
    initial_lr = learning_rate  # Initial learning rate
    drop = 0.5  # Learning rate will be multiplied by this factor
    epochs_drop = 2  # Number of epochs after which the learning rate will be dropped
    lr = initial_lr * tf.math.pow(drop, tf.math.floor((1 + epoch) / epochs_drop))
    return lr
```

In [249…

```
def create_cnn_model(input_shape, layer_sizes, filter_size, pool_size, hidden_size, dropout_rate, lea
    """
    Creates a Convolutional Neural Network (CNN) model for facial keypoint detection.

    Parameters:
    - input_shape (tuple): Shape of the input images (e.g., (96, 96, 1) for grayscale images).
    - layer_sizes (list): List containing the number of filters for each convolutional layer.
    - filter_size (int): Size of the convolutional filters.
    - pool_size (int or tuple): Size of the max pooling window.
    - hidden_size (int): Number of neurons in the dense hidden layer.
    - dropout_rate (float): Dropout rate to prevent overfitting.
    - learning_rate (float): Learning rate for the Adam optimizer.

    Returns:
    - tf.keras.models.Sequential: Compiled CNN model for facial keypoint detection.
    """
    model = models.Sequential()

    # Convolutional layers with Batch Normalization
    model.add(layers.Conv2D(layer_sizes[0], filter_size, activation='relu', input_shape=input_shape))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size))

    model.add(layers.Conv2D(layer_sizes[1], filter_size, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size))

    model.add(layers.Conv2D(layer_sizes[2], filter_size, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size))

    # Flatten layer
    model.add(layers.Flatten())

    # Dense layers
    model.add(layers.Dense(hidden_size, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(dropout_rate))

    model.add(layers.Dense(30, activation='linear'))

    # Compile the model with mean squared error loss
    opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=opt,
                  loss='mean_squared_error',
                  metrics=['mae', tf.keras.metrics.RootMeanSquaredError()])

    return model
```

In [250…

```
dataset_train, dataset_val, dataset_test = create_batches(BATCH_SIZE, df_train_clean, dataset_trainin
```

In [251…

```
# Set the input shape
input_shape = (96, 96, 1)

layer_sizes = (32, 64, 128)
filter_size = (3,3)
pool_size = (2,2)
hidden_size = 256
dropout_rate = 0.5
learning_rate = 0.005

# Define the Learning Rate Scheduler callback
lr_scheduler = LearningRateScheduler(step_decay)
```

```
# Create the model
model = create_cnn_model(input_shape, layer_sizes, filter_size, pool_size, hidden_size, dropout_rate,

# Display the model architecture
model.summary()
```

```
Model: "sequential_110"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_330 (Conv2D)         (None, 94, 94, 32)        320

 batch_normalization_220 (Ba  (None, 94, 94, 32)       128
 tchNormalization)

 max_pooling2d_330 (MaxPooli  (None, 47, 47, 32)       0
 ng2D)

 conv2d_331 (Conv2D)         (None, 45, 45, 64)        18496

 batch_normalization_221 (Ba  (None, 45, 45, 64)       256
 tchNormalization)

 max_pooling2d_331 (MaxPooli  (None, 22, 22, 64)       0
 ng2D)

 conv2d_332 (Conv2D)         (None, 20, 20, 128)       73856

 batch_normalization_222 (Ba  (None, 20, 20, 128)      512
 tchNormalization)

 max_pooling2d_332 (MaxPooli  (None, 10, 10, 128)      0
 ng2D)

 flatten_110 (Flatten)       (None, 12800)             0

 dense_220 (Dense)           (None, 256)               3277056

 batch_normalization_223 (Ba  (None, 256)              1024
 tchNormalization)

 dropout_110 (Dropout)       (None, 256)               0

 dense_221 (Dense)           (None, 30)                7710

=================================================================
Total params: 3,379,358
Trainable params: 3,378,398
Non-trainable params: 960
_________________________________________________________________
```

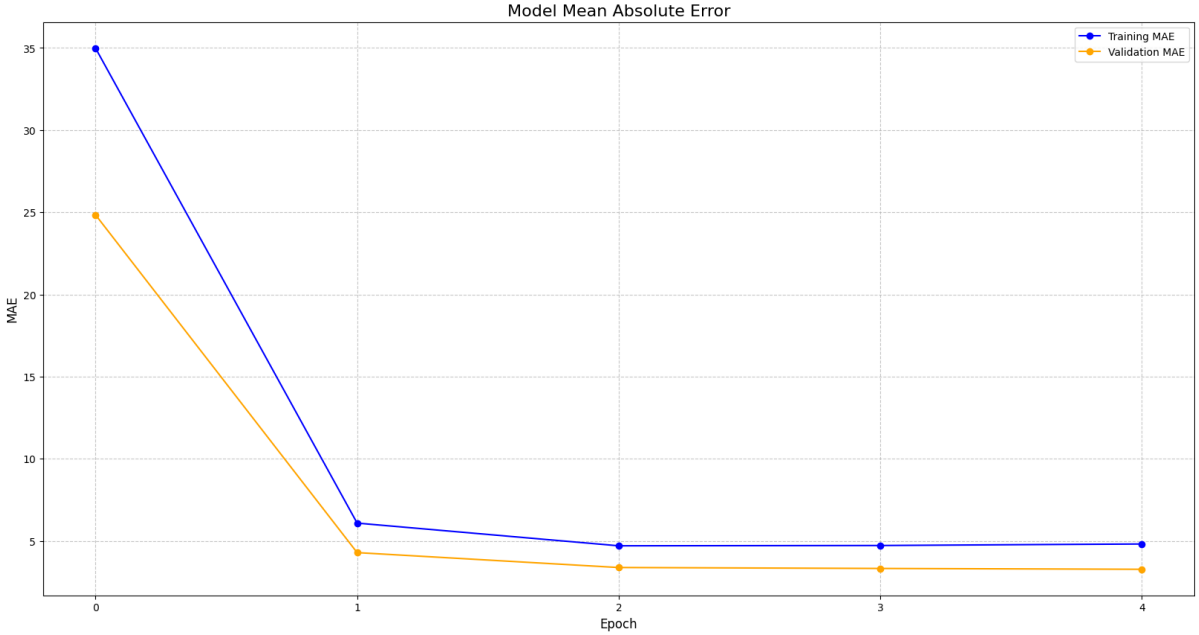
In [252…

```
training_history = model.fit(dataset_train, batch_size=BATCH_SIZE, epochs=EPOCHS, validation_data=data
```

```
Epoch 1/5
107/107 [==============================] - 19s 167ms/step - loss: 1685.5444 - mae: 34.9881 - root_mean
_squared_error: 41.0554 - val_loss: 996.2111 - val_mae: 24.8435 - val_root_mean_squared_error: 31.5628
- lr: 0.0050
Epoch 2/5
107/107 [==============================] - 18s 167ms/step - loss: 69.8337 - mae: 6.0864 - root_mean_sq
uared_error: 8.3567 - val_loss: 33.3205 - val_mae: 4.2906 - val_root_mean_squared_error: 5.7724 - lr:
0.0025
Epoch 3/5
107/107 [==============================] - 18s 167ms/step - loss: 39.2414 - mae: 4.7054 - root_mean_sq
uared_error: 6.2643 - val_loss: 20.4410 - val_mae: 3.3831 - val_root_mean_squared_error: 4.5212 - lr:
0.0012
Epoch 4/5
107/107 [==============================] - 18s 167ms/step - loss: 39.7214 - mae: 4.7237 - root_mean_sq
uared_error: 6.3025 - val_loss: 19.6488 - val_mae: 3.3275 - val_root_mean_squared_error: 4.4327 - lr:
3.1250e-04
Epoch 5/5
107/107 [==============================] - 18s 168ms/step - loss: 41.4846 - mae: 4.8161 - root_mean_sq
uared_error: 6.4409 - val_loss: 18.9556 - val_mae: 3.2750 - val_root_mean_squared_error: 4.3538 - lr:
7.8125e-05
```

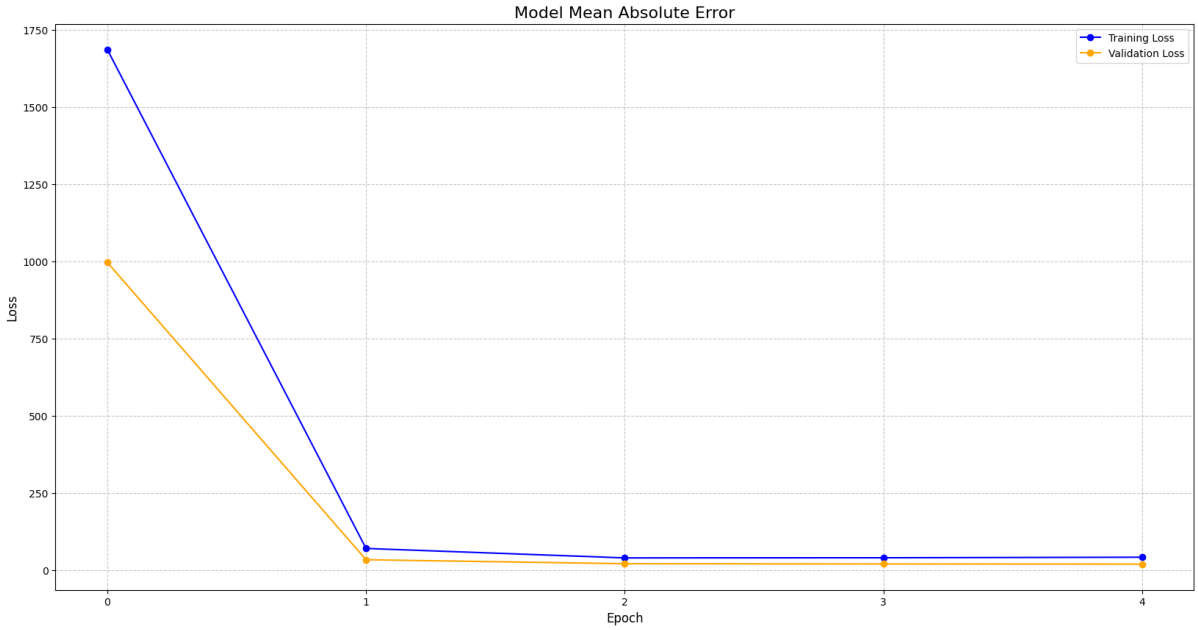
From the dynamic fitting progress bar, it's apparent that the model is making consistent improvements with each epoch. To substantiate this observation, we will create a visual representation in the form of a plot, illustrating the evolution of loss and accuracy metrics for both the training and validation datasets.

In [253…
```
# Extract training history
train_mae = training_history.history['mae']
val_mae = training_history.history['val_mae']

# Set up a more visually appealing plot
plt.figure(figsize=(20, 10))
plt.plot(train_mae, label='Training MAE', color='blue', marker='o')
plt.plot(val_mae, label='Validation MAE', color='orange', marker='o')

# Set plot title and labels
plt.title('Model Mean Absolute Error', fontsize=16)
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('MAE', fontsize=12)

# Add legend and adjust its position
plt.legend(loc='upper right')

# Add grid lines for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Set x-axis to start from 1 and display integers
plt.xticks(range(0, EPOCHS))

# Show the plot
plt.show()
```



In [254…
```
# Extract training history
train_loss = training_history.history['loss']
val_loss = training_history.history['val_loss']

# Set up a more visually appealing plot
plt.figure(figsize=(20, 10))
plt.plot(train_loss, label='Training Loss', color='blue', marker='o')
plt.plot(val_loss, label='Validation Loss', color='orange', marker='o')

# Set plot title and labels
plt.title('Model Mean Absolute Error', fontsize=16)
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)

# Add legend and adjust its position
plt.legend(loc='upper right')

# Add grid lines for better readability
```

```
plt.grid(True, linestyle='--', alpha=0.7)

# Set x-axis to start from 1 and display integers
plt.xticks(range(0, EPOCHS))

# Show the plot
plt.show()
```



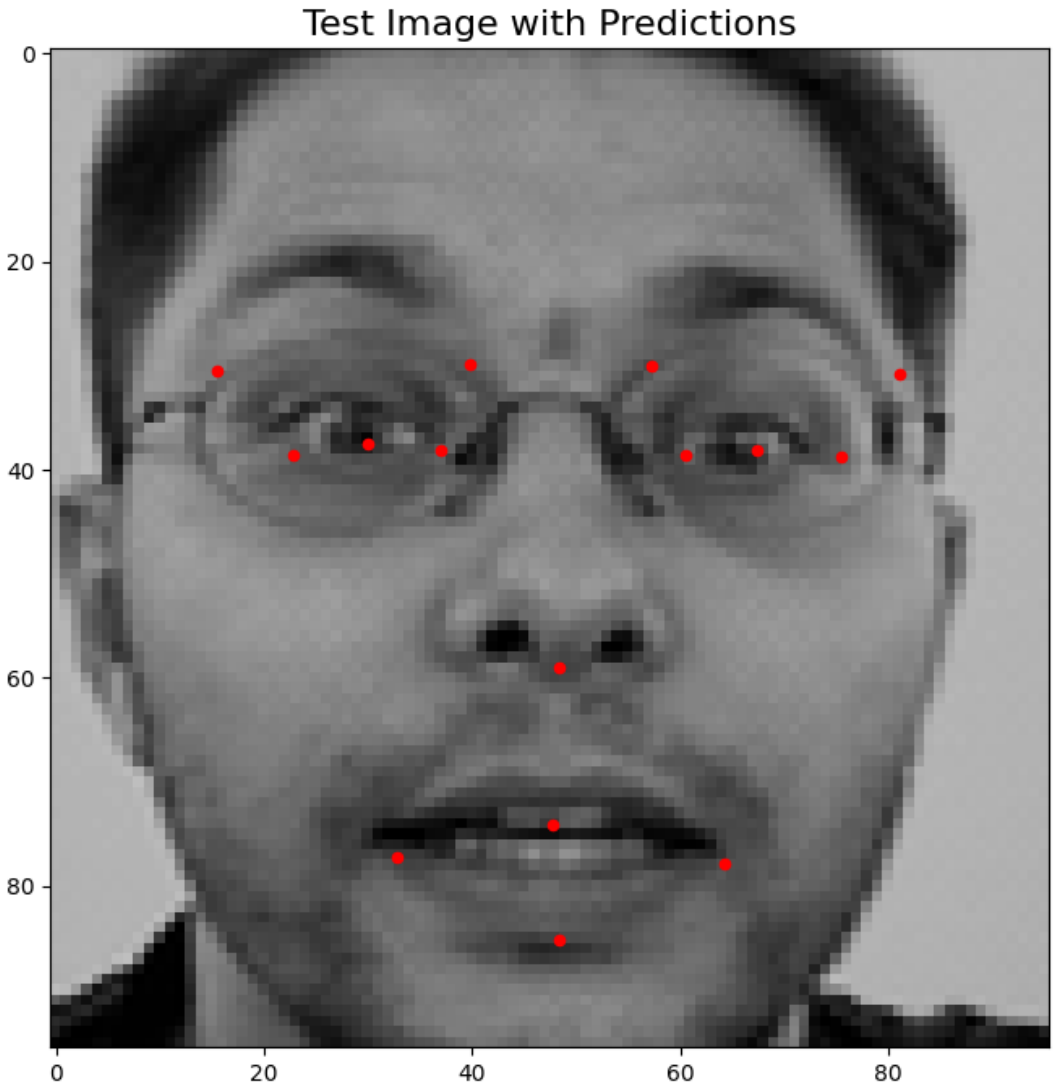
We observe from the plotted graphs that both the loss and mean absolute error (our chosen metric) consistently improve for both the training and validation datasets, indicating positive progress. This lack of overfitting throughout the training process is an encouraging outcome.

To conclude, we will generate a plot showcasing an example image from the test set alongside the predicted facial feature positions. This visual verification affirms that our initial model attempt performs commendably. Nonetheless, recognizing the potential for enhancement, we will endeavor to optimize our results in the upcoming hyperparameter optimization phase by adjusting various training and model parameters.

In [255…

```
# Extract a single test image and its prediction
test_image = list(dataset_test.take(1).as_numpy_iterator())[0][0]
test_prediction = model.predict(dataset_test.take(1))[0]

# Set up a more visually appealing plot
fig, ax = plt.subplots(figsize=(8, 8))

# Display the test image
ax.imshow(test_image, cmap='gray', vmin=0, vmax=255)

# Scatter plot for predictions
ax.scatter(test_prediction[0::2], test_prediction[1::2], s=20, c='red', marker='o')

# Set plot title and labels
ax.set_title('Test Image with Predictions', fontsize=16)

# Show the plot
plt.show()
```

```
WARNING:tensorflow:5 out of the last 14 calls to <function Model.make_predict_function.<locals>.predic
t_function at 0x000001A802AF3E20> triggered tf.function retracing. Tracing is expensive and the excess
ive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing ten
sors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your
@tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoi
d unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controllin
g_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 123ms/step
```



Hyperparameter Optimization

We can leverage the previously defined generic functions for both the model and data creation processes as the foundation for our parameter optimization steps. These functions are designed to accommodate varying input parameters, which will be explored in the upcoming optimization process.

Rather than employing a grid search, we opt for a more nuanced approach to better discern the effects of different parameters. Although this method may not guarantee the absolute best solution, the goal is to gain insights and understanding through the learning process, which aligns with the objectives of a course project.

Additionally, to align with the evaluation metric used by the Kaggle scoring algorithm, we will transition to using the root mean squared error as the metric for assessing model performance in our pursuit of finding the optimal model for the competition.

In [257…

```
def hyp_tun(BATCH_SIZE, layer_sizes, filter_size, pool_size, hidden_size, dropout_rate, learning_rate
    """
    Perform hyperparameter tuning for a CNN model.

    Parameters:
    - BATCH_SIZE (int): Batch size for training.
    - layer_sizes (tuple): Tuple containing the number of filters for each convolutional layer.
    - filter_size (tuple): Tuple containing the filter size for each convolutional layer.
    - pool_size (tuple): Tuple containing the pool size for each max pooling layer.
    - hidden_size (int): Number of units in the dense hidden layer.
    - dropout_rate (float): Dropout rate for regularization.
    - learning_rate (float): Learning rate for the optimizer.

    Returns:
    - history (tf.keras.callbacks.History): Training history containing loss and metrics over epochs.
```

```
    """
    # Create training, validation, and testing datasets
    dataset_train, dataset_val, dataset_test = create_batches(BATCH_SIZE, df_train_clean, dataset_tra

    # Create the model
    model = create_cnn_model(input_shape, layer_sizes, filter_size, pool_size, hidden_size, dropout_r

    # Train the model and store the training history
    history = model.fit(dataset_train, batch_size=BATCH_SIZE, epochs=EPOCHS, verbose= False, validati

    return history
```

In [258…

```
def hyp_tun_plot(hists, label, sizes):
    # Set up a more visually appealing plot
    plt.figure(figsize=(20, 10))

    # Plot training and validation RMSE for each configuration
    for i, hist in enumerate(hists):
        plt.plot(hist.history['root_mean_squared_error'], label=f'Train {label} {sizes[i]}', linestyl
        plt.plot(hist.history['val_root_mean_squared_error'], label=f'Validation {label} {sizes[i]}',

    # Set plot title and labels
    plt.title('Model Root Mean Squared Error', fontsize=16)
    plt.xlabel('Epoch', fontsize=12)
    plt.ylabel('RMSE', fontsize=12)

    # Set y-axis limits to 0 and 10
    plt.ylim(0, 10)

    # Add legend and adjust its position
    plt.legend(loc='upper right')

    # Add grid lines for better readability
    plt.grid(True, linestyle='--', alpha=0.7)

    # Set x-axis to start from 1 and display integers
    plt.xticks(range(0, EPOCHS))

    # Show the plot
    plt.show()
```

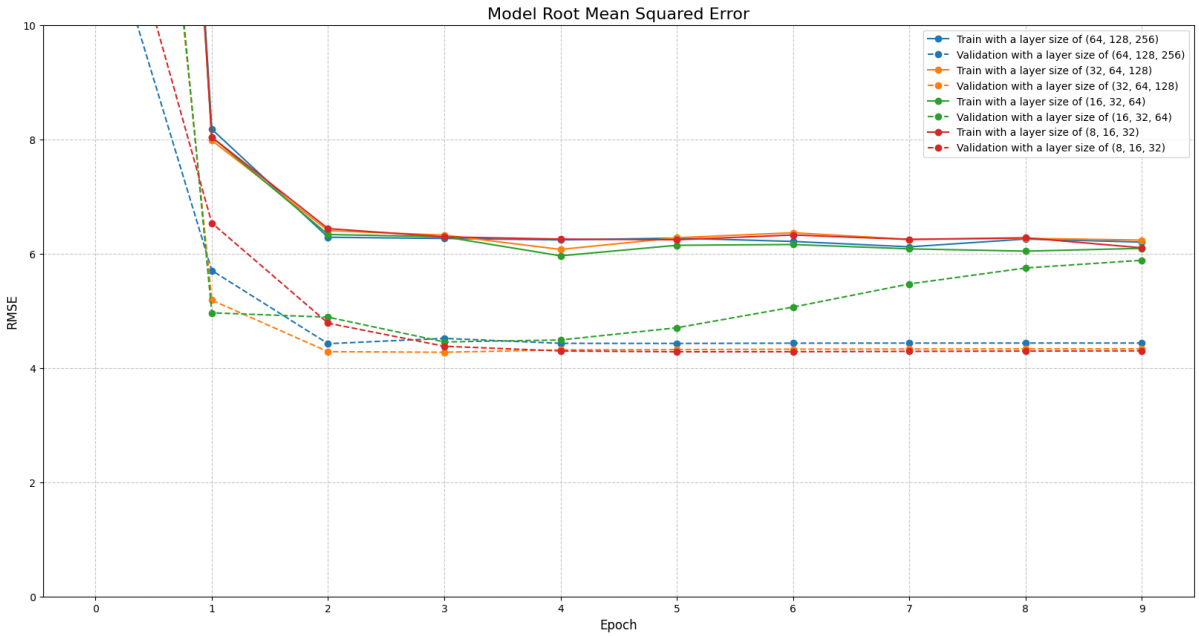
Initially, I will experiment with varying the number of layers in each convolutional step, starting with a baseline of 32, 64, and 128.

In [259…

```
var_layer_sizes = [(64, 128, 256), (32, 64, 128), (16, 32, 64), (8, 16, 32)]
hists = []

for sizes in var_layer_sizes:
    hists.append(hyp_tun(BATCH_SIZE, layer_sizes = sizes, filter_size = (3, 3), pool_size = (2, 2), h
```

In [260…

```
hyp_tun_plot(hists, 'with a layer size of', var_layer_sizes)
```



The results indicate that the medium-sized model performs just as well as the larger one, demonstrating comparable generalization abilities, as evidenced by the low root mean squared error (RMSE) scores on the validation data. Consequently, we will proceed with this model, characterized by convolution layer sizes of 16, 32, and 64, respectively, to minimize resource utilization.

Moving forward, we will explore variations in the size of the hidden dense layer of the model.

In [208…

```
var_dense = [256,128,64,32,16]
hists2 = []
for sizes in var_dense:
    hists2.append(hyp_tun(BATCH_SIZE, layer_sizes = (32, 64, 128), filter_size = (3, 3), pool_size =
```

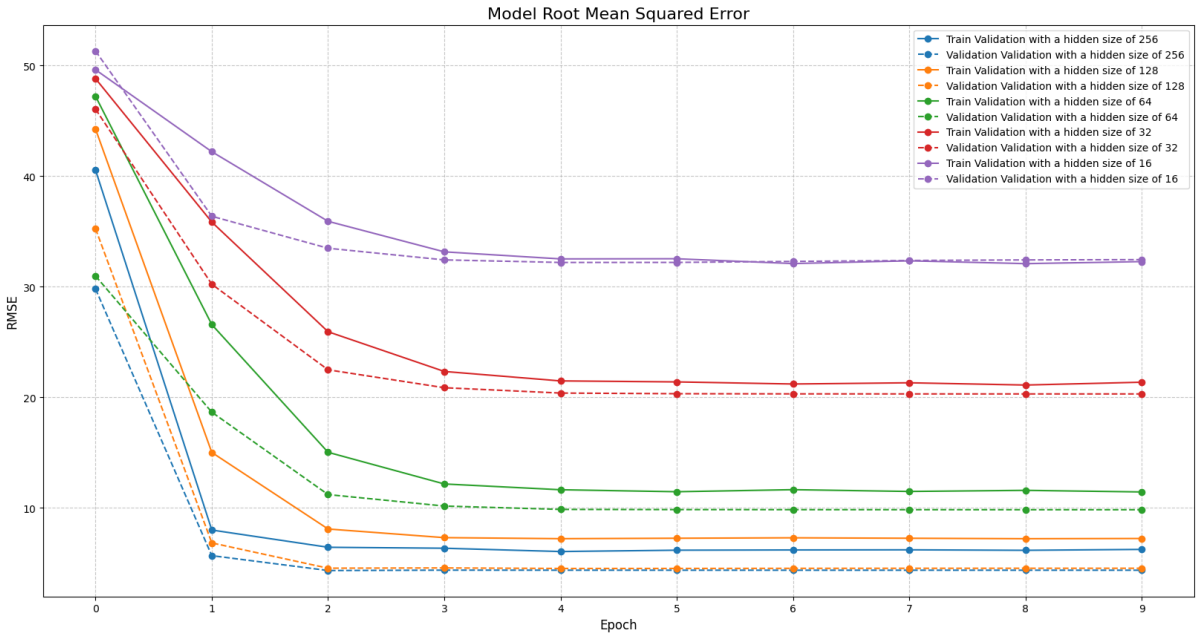
```
Epoch 1/10
107/107 [==============================] - 18s 158ms/step - loss: 1644.2130 - mae: 34.3774 - root_mean
_squared_error: 40.5489 - val_loss: 887.9245 - val_mae: 24.4604 - val_root_mean_squared_error: 29.7981
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 159ms/step - loss: 64.0721 - mae: 5.8877 - root_mean_sq
uared_error: 8.0045 - val_loss: 32.3625 - val_mae: 4.3898 - val_root_mean_squared_error: 5.6888 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 159ms/step - loss: 41.5628 - mae: 4.8078 - root_mean_sq
uared_error: 6.4469 - val_loss: 18.8734 - val_mae: 3.2698 - val_root_mean_squared_error: 4.3443 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 161ms/step - loss: 40.4356 - mae: 4.7338 - root_mean_sq
uared_error: 6.3589 - val_loss: 19.1746 - val_mae: 3.2937 - val_root_mean_squared_error: 4.3789 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 161ms/step - loss: 36.7243 - mae: 4.5451 - root_mean_sq
uared_error: 6.0601 - val_loss: 19.1378 - val_mae: 3.2909 - val_root_mean_squared_error: 4.3747 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 161ms/step - loss: 38.2284 - mae: 4.6381 - root_mean_sq
uared_error: 6.1829 - val_loss: 19.1279 - val_mae: 3.2898 - val_root_mean_squared_error: 4.3735 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 159ms/step - loss: 38.4880 - mae: 4.6274 - root_mean_sq
uared_error: 6.2039 - val_loss: 19.1195 - val_mae: 3.2889 - val_root_mean_squared_error: 4.3726 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 17s 158ms/step - loss: 38.6281 - mae: 4.6882 - root_mean_sq
uared_error: 6.2152 - val_loss: 19.1202 - val_mae: 3.2891 - val_root_mean_squared_error: 4.3727 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 158ms/step - loss: 38.1083 - mae: 4.6387 - root_mean_sq
uared_error: 6.1732 - val_loss: 19.1211 - val_mae: 3.2892 - val_root_mean_squared_error: 4.3728 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 161ms/step - loss: 39.0808 - mae: 4.6506 - root_mean_sq
uared_error: 6.2515 - val_loss: 19.1216 - val_mae: 3.2892 - val_root_mean_squared_error: 4.3728 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 157ms/step - loss: 1958.7493 - mae: 39.3947 - root_mean
_squared_error: 44.2578 - val_loss: 1242.5624 - val_mae: 28.4498 - val_root_mean_squared_error: 35.250
0 - lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 155ms/step - loss: 225.6145 - mae: 10.5268 - root_mean_
squared_error: 15.0205 - val_loss: 46.7924 - val_mae: 5.0288 - val_root_mean_squared_error: 6.8405 - l
r: 0.0025
Epoch 3/10
107/107 [==============================] - 17s 155ms/step - loss: 65.5906 - mae: 5.9878 - root_mean_sq
uared_error: 8.0988 - val_loss: 20.7368 - val_mae: 3.4482 - val_root_mean_squared_error: 4.5538 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 16s 154ms/step - loss: 53.5279 - mae: 5.4619 - root_mean_sq
uared_error: 7.3163 - val_loss: 20.9441 - val_mae: 3.4431 - val_root_mean_squared_error: 4.5765 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 16s 154ms/step - loss: 52.1901 - mae: 5.4128 - root_mean_sq
uared_error: 7.2243 - val_loss: 20.5014 - val_mae: 3.4002 - val_root_mean_squared_error: 4.5278 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 156ms/step - loss: 52.7619 - mae: 5.4355 - root_mean_sq
uared_error: 7.2637 - val_loss: 20.5232 - val_mae: 3.4001 - val_root_mean_squared_error: 4.5303 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 155ms/step - loss: 53.3166 - mae: 5.4985 - root_mean_sq
uared_error: 7.3018 - val_loss: 20.6007 - val_mae: 3.4050 - val_root_mean_squared_error: 4.5388 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 16s 154ms/step - loss: 52.7365 - mae: 5.4640 - root_mean_sq
uared_error: 7.2620 - val_loss: 20.6496 - val_mae: 3.4083 - val_root_mean_squared_error: 4.5442 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 155ms/step - loss: 52.1533 - mae: 5.4072 - root_mean_sq
```

```
uared_error: 7.2217 - val_loss: 20.6684 - val_mae: 3.4096 - val_root_mean_squared_error: 4.5463 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 156ms/step - loss: 52.4196 - mae: 5.4554 - root_mean_sq
uared_error: 7.2401 - val_loss: 20.6750 - val_mae: 3.4100 - val_root_mean_squared_error: 4.5470 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 17s 155ms/step - loss: 2230.3215 - mae: 43.1090 - root_mean
_squared_error: 47.2263 - val_loss: 959.6790 - val_mae: 24.8942 - val_root_mean_squared_error: 30.9787
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 16s 154ms/step - loss: 706.5729 - mae: 20.6168 - root_mean_
squared_error: 26.5814 - val_loss: 348.2813 - val_mae: 14.3548 - val_root_mean_squared_error: 18.6623
- lr: 0.0025
Epoch 3/10
107/107 [==============================] - 17s 155ms/step - loss: 226.0204 - mae: 10.8314 - root_mean_
squared_error: 15.0340 - val_loss: 125.6970 - val_mae: 7.7723 - val_root_mean_squared_error: 11.2115 -
lr: 0.0012
Epoch 4/10
107/107 [==============================] - 16s 152ms/step - loss: 148.1241 - mae: 8.8401 - root_mean_s
quared_error: 12.1706 - val_loss: 103.5018 - val_mae: 7.0428 - val_root_mean_squared_error: 10.1736 -
lr: 3.1250e-04
Epoch 5/10
107/107 [==============================] - 16s 151ms/step - loss: 135.6690 - mae: 8.4313 - root_mean_s
quared_error: 11.6477 - val_loss: 97.4073 - val_mae: 6.8364 - val_root_mean_squared_error: 9.8695 - l
r: 7.8125e-05
Epoch 6/10
107/107 [==============================] - 16s 150ms/step - loss: 131.5035 - mae: 8.3282 - root_mean_s
quared_error: 11.4675 - val_loss: 97.0876 - val_mae: 6.8290 - val_root_mean_squared_error: 9.8533 - l
r: 9.7656e-06
Epoch 7/10
107/107 [==============================] - 16s 150ms/step - loss: 135.8266 - mae: 8.4725 - root_mean_s
quared_error: 11.6545 - val_loss: 96.9691 - val_mae: 6.8266 - val_root_mean_squared_error: 9.8473 - l
r: 1.2207e-06
Epoch 8/10
107/107 [==============================] - 16s 150ms/step - loss: 132.1030 - mae: 8.3478 - root_mean_s
quared_error: 11.4936 - val_loss: 96.9140 - val_mae: 6.8245 - val_root_mean_squared_error: 9.8445 - l
r: 7.6294e-08
Epoch 9/10
107/107 [==============================] - 16s 151ms/step - loss: 134.4094 - mae: 8.3976 - root_mean_s
quared_error: 11.5935 - val_loss: 96.8675 - val_mae: 6.8221 - val_root_mean_squared_error: 9.8421 - l
r: 4.7684e-09
Epoch 10/10
107/107 [==============================] - 16s 150ms/step - loss: 131.0416 - mae: 8.3176 - root_mean_s
quared_error: 11.4473 - val_loss: 96.8488 - val_mae: 6.8209 - val_root_mean_squared_error: 9.8412 - l
r: 1.4901e-10
Epoch 1/10
107/107 [==============================] - 17s 149ms/step - loss: 2383.4556 - mae: 44.9978 - root_mean
_squared_error: 48.8206 - val_loss: 2121.5701 - val_mae: 41.1501 - val_root_mean_squared_error: 46.060
5 - lr: 0.0050
Epoch 2/10
107/107 [==============================] - 16s 147ms/step - loss: 1285.3716 - mae: 30.6515 - root_mean
_squared_error: 35.8521 - val_loss: 913.8417 - val_mae: 24.6478 - val_root_mean_squared_error: 30.2298
- lr: 0.0025
Epoch 3/10
107/107 [==============================] - 16s 149ms/step - loss: 672.6605 - mae: 20.1442 - root_mean_
squared_error: 25.9357 - val_loss: 505.9546 - val_mae: 16.7893 - val_root_mean_squared_error: 22.4934
- lr: 0.0012
Epoch 4/10
107/107 [==============================] - 15s 144ms/step - loss: 499.2408 - mae: 16.7766 - root_mean_
squared_error: 22.3437 - val_loss: 435.7116 - val_mae: 15.3063 - val_root_mean_squared_error: 20.8737
- lr: 3.1250e-04
Epoch 5/10
107/107 [==============================] - 15s 145ms/step - loss: 461.7571 - mae: 16.0083 - root_mean_
squared_error: 21.4885 - val_loss: 415.7738 - val_mae: 14.8584 - val_root_mean_squared_error: 20.3905
- lr: 7.8125e-05
Epoch 6/10
107/107 [==============================] - 16s 145ms/step - loss: 457.9557 - mae: 15.9470 - root_mean_
squared_error: 21.3999 - val_loss: 413.2907 - val_mae: 14.8061 - val_root_mean_squared_error: 20.3296
- lr: 9.7656e-06
Epoch 7/10
107/107 [==============================] - 15s 144ms/step - loss: 449.8339 - mae: 15.7553 - root_mean_
squared_error: 21.2093 - val_loss: 412.7591 - val_mae: 14.7950 - val_root_mean_squared_error: 20.3165
- lr: 1.2207e-06
```

```
Epoch 8/10
107/107 [==============================] - 15s 143ms/step - loss: 454.5213 - mae: 15.8789 - root_mean_
squared_error: 21.3195 - val_loss: 412.6107 - val_mae: 14.7920 - val_root_mean_squared_error: 20.3128
- lr: 7.6294e-08
Epoch 9/10
107/107 [==============================] - 15s 145ms/step - loss: 445.8932 - mae: 15.6949 - root_mean_
squared_error: 21.1162 - val_loss: 412.5652 - val_mae: 14.7911 - val_root_mean_squared_error: 20.3117
- lr: 4.7684e-09
Epoch 10/10
107/107 [==============================] - 15s 145ms/step - loss: 456.7375 - mae: 15.8862 - root_mean_
squared_error: 21.3714 - val_loss: 412.5493 - val_mae: 14.7908 - val_root_mean_squared_error: 20.3113
- lr: 1.4901e-10
Epoch 1/10
107/107 [==============================] - 17s 146ms/step - loss: 2462.5190 - mae: 45.9169 - root_mean
_squared_error: 49.6238 - val_loss: 2633.7986 - val_mae: 47.5012 - val_root_mean_squared_error: 51.320
5 - lr: 0.0050
Epoch 2/10
107/107 [==============================] - 15s 144ms/step - loss: 1782.1058 - mae: 37.7400 - root_mean
_squared_error: 42.2150 - val_loss: 1323.3729 - val_mae: 31.4576 - val_root_mean_squared_error: 36.378
2 - lr: 0.0025
Epoch 3/10
107/107 [==============================] - 16s 146ms/step - loss: 1290.3251 - mae: 30.7029 - root_mean
_squared_error: 35.9211 - val_loss: 1121.3112 - val_mae: 28.2948 - val_root_mean_squared_error: 33.486
0 - lr: 0.0012
Epoch 4/10
107/107 [==============================] - 16s 147ms/step - loss: 1099.6050 - mae: 27.6276 - root_mean
_squared_error: 33.1603 - val_loss: 1051.7673 - val_mae: 27.1381 - val_root_mean_squared_error: 32.431
0 - lr: 3.1250e-04
Epoch 5/10
107/107 [==============================] - 16s 147ms/step - loss: 1057.5890 - mae: 26.8518 - root_mean
_squared_error: 32.5206 - val_loss: 1036.9604 - val_mae: 26.8780 - val_root_mean_squared_error: 32.201
9 - lr: 7.8125e-05
Epoch 6/10
107/107 [==============================] - 16s 146ms/step - loss: 1058.5251 - mae: 26.9302 - root_mean
_squared_error: 32.5350 - val_loss: 1037.2434 - val_mae: 26.8664 - val_root_mean_squared_error: 32.206
3 - lr: 9.7656e-06
Epoch 7/10
107/107 [==============================] - 16s 149ms/step - loss: 1030.3881 - mae: 26.4320 - root_mean
_squared_error: 32.0997 - val_loss: 1042.9132 - val_mae: 26.9407 - val_root_mean_squared_error: 32.294
2 - lr: 1.2207e-06
Epoch 8/10
107/107 [==============================] - 16s 151ms/step - loss: 1046.4266 - mae: 26.6832 - root_mean
_squared_error: 32.3485 - val_loss: 1048.3093 - val_mae: 27.0136 - val_root_mean_squared_error: 32.377
6 - lr: 7.6294e-08
Epoch 9/10
107/107 [==============================] - 16s 151ms/step - loss: 1029.9547 - mae: 26.4275 - root_mean
_squared_error: 32.0929 - val_loss: 1051.6600 - val_mae: 27.0599 - val_root_mean_squared_error: 32.429
3 - lr: 4.7684e-09
Epoch 10/10
107/107 [==============================] - 16s 152ms/step - loss: 1041.7711 - mae: 26.6212 - root_mean
_squared_error: 32.2765 - val_loss: 1053.5725 - val_mae: 27.0875 - val_root_mean_squared_error: 32.458
8 - lr: 1.4901e-10
```

In [209…

```
hyp_tun_plot(hists2, 'Validation with a hidden size of', var_dense)
```



Based on the results, we opt to retain the hidden dense layer with 256 neurons, as further increasing the layer size does not yield improvements in validation results. Conversely, smaller layer sizes result in a decrease in predictive performance, as evidenced by higher root mean squared error (RMSE) values. In the next stage, we will investigate whether adjusting the learning rate for the Adam optimizer to be faster or slower enhances the overall performance of the model.

In [210…

```
var_lr = [0.0001,0.0005,0.001,0.005, 0.01]
hists3 = []
for lrs in var_lr:
    hists3.append(hyp_tun(BATCH_SIZE, layer_sizes = (32, 64, 128), filter_size = (3, 3), pool_size =
```

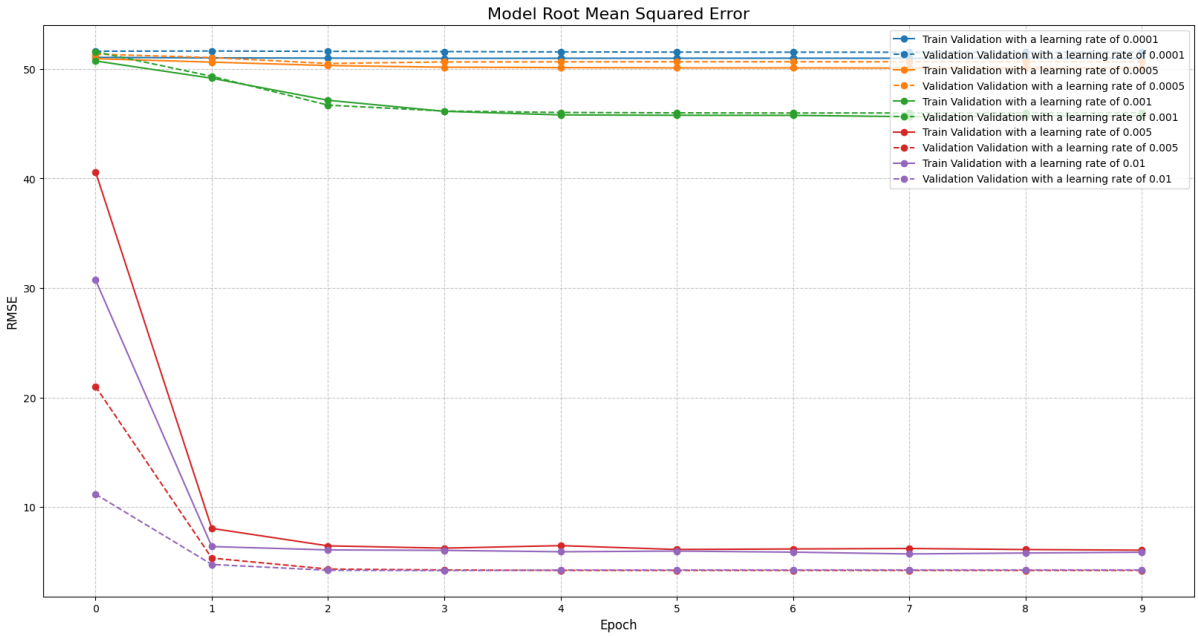
```
Epoch 1/10
107/107 [==============================] - 18s 165ms/step - loss: 2606.0002 - mae: 47.4531 - root_mean
_squared_error: 51.0490 - val_loss: 2664.0376 - val_mae: 47.9013 - val_root_mean_squared_error: 51.614
3 - lr: 1.0000e-04
Epoch 2/10
107/107 [==============================] - 17s 163ms/step - loss: 2601.6245 - mae: 47.4278 - root_mean
_squared_error: 51.0061 - val_loss: 2665.9929 - val_mae: 47.9522 - val_root_mean_squared_error: 51.633
3 - lr: 5.0000e-05
Epoch 3/10
107/107 [==============================] - 17s 160ms/step - loss: 2599.3435 - mae: 47.4082 - root_mean
_squared_error: 50.9838 - val_loss: 2662.7534 - val_mae: 47.9342 - val_root_mean_squared_error: 51.601
9 - lr: 2.5000e-05
Epoch 4/10
107/107 [==============================] - 17s 163ms/step - loss: 2597.3711 - mae: 47.3915 - root_mean
_squared_error: 50.9644 - val_loss: 2660.3481 - val_mae: 47.9158 - val_root_mean_squared_error: 51.578
6 - lr: 6.2500e-06
Epoch 5/10
107/107 [==============================] - 18s 167ms/step - loss: 2598.0562 - mae: 47.3928 - root_mean
_squared_error: 50.9711 - val_loss: 2657.4153 - val_mae: 47.8849 - val_root_mean_squared_error: 51.550
1 - lr: 1.5625e-06
Epoch 6/10
107/107 [==============================] - 18s 166ms/step - loss: 2598.4551 - mae: 47.4023 - root_mean
_squared_error: 50.9750 - val_loss: 2655.7109 - val_mae: 47.8661 - val_root_mean_squared_error: 51.533
6 - lr: 1.9531e-07
Epoch 7/10
107/107 [==============================] - 17s 162ms/step - loss: 2598.5923 - mae: 47.4016 - root_mean
_squared_error: 50.9764 - val_loss: 2654.9048 - val_mae: 47.8555 - val_root_mean_squared_error: 51.525
8 - lr: 2.4414e-08
Epoch 8/10
107/107 [==============================] - 18s 166ms/step - loss: 2598.2080 - mae: 47.3995 - root_mean
_squared_error: 50.9726 - val_loss: 2654.6404 - val_mae: 47.8510 - val_root_mean_squared_error: 51.523
2 - lr: 1.5259e-09
Epoch 9/10
107/107 [==============================] - 18s 168ms/step - loss: 2598.1033 - mae: 47.3978 - root_mean
_squared_error: 50.9716 - val_loss: 2654.5552 - val_mae: 47.8492 - val_root_mean_squared_error: 51.522
4 - lr: 9.5367e-11
Epoch 10/10
107/107 [==============================] - 18s 168ms/step - loss: 2597.1445 - mae: 47.3880 - root_mean
_squared_error: 50.9622 - val_loss: 2654.5293 - val_mae: 47.8486 - val_root_mean_squared_error: 51.522
1 - lr: 2.9802e-12
Epoch 1/10
107/107 [==============================] - 19s 170ms/step - loss: 2594.1980 - mae: 47.3484 - root_mean
_squared_error: 50.9333 - val_loss: 2637.0591 - val_mae: 47.7097 - val_root_mean_squared_error: 51.352
3 - lr: 5.0000e-04
Epoch 2/10
107/107 [==============================] - 18s 169ms/step - loss: 2562.0122 - mae: 47.0238 - root_mean
_squared_error: 50.6163 - val_loss: 2604.0830 - val_mae: 47.3773 - val_root_mean_squared_error: 51.030
2 - lr: 2.5000e-04
Epoch 3/10
107/107 [==============================] - 18s 169ms/step - loss: 2530.7302 - mae: 46.7048 - root_mean
_squared_error: 50.3064 - val_loss: 2549.8657 - val_mae: 46.7928 - val_root_mean_squared_error: 50.496
2 - lr: 1.2500e-04
Epoch 4/10
107/107 [==============================] - 18s 171ms/step - loss: 2515.7058 - mae: 46.5554 - root_mean
_squared_error: 50.1568 - val_loss: 2563.8328 - val_mae: 46.9428 - val_root_mean_squared_error: 50.634
3 - lr: 3.1250e-05
Epoch 5/10
107/107 [==============================] - 18s 168ms/step - loss: 2511.7390 - mae: 46.5098 - root_mean
_squared_error: 50.1172 - val_loss: 2565.0193 - val_mae: 46.9526 - val_root_mean_squared_error: 50.646
0 - lr: 7.8125e-06
Epoch 6/10
107/107 [==============================] - 18s 170ms/step - loss: 2509.3357 - mae: 46.4792 - root_mean
_squared_error: 50.0933 - val_loss: 2565.8298 - val_mae: 46.9582 - val_root_mean_squared_error: 50.654
0 - lr: 9.7656e-07
Epoch 7/10
107/107 [==============================] - 18s 167ms/step - loss: 2509.4746 - mae: 46.4849 - root_mean
_squared_error: 50.0947 - val_loss: 2565.8423 - val_mae: 46.9544 - val_root_mean_squared_error: 50.654
1 - lr: 1.2207e-07
Epoch 8/10
107/107 [==============================] - 18s 168ms/step - loss: 2507.7156 - mae: 46.4664 - root_mean
_squared_error: 50.0771 - val_loss: 2565.4961 - val_mae: 46.9450 - val_root_mean_squared_error: 50.650
7 - lr: 7.6294e-09
Epoch 9/10
107/107 [==============================] - 18s 167ms/step - loss: 2509.1245 - mae: 46.4806 - root_mean
```

```
_squared_error: 50.0912 - val_loss: 2565.1365 - val_mae: 46.9334 - val_root_mean_squared_error: 50.647
2 - lr: 4.7684e-10
Epoch 10/10
107/107 [==============================] - 18s 167ms/step - loss: 2508.1682 - mae: 46.4687 - root_mean
_squared_error: 50.0816 - val_loss: 2564.8931 - val_mae: 46.9254 - val_root_mean_squared_error: 50.644
8 - lr: 1.4901e-11
Epoch 1/10
107/107 [==============================] - 18s 166ms/step - loss: 2573.1069 - mae: 47.1333 - root_mean
_squared_error: 50.7258 - val_loss: 2659.0562 - val_mae: 47.9583 - val_root_mean_squared_error: 51.566
0 - lr: 0.0010
Epoch 2/10
107/107 [==============================] - 17s 160ms/step - loss: 2415.2874 - mae: 45.4911 - root_mean
_squared_error: 49.1456 - val_loss: 2432.8215 - val_mae: 45.6331 - val_root_mean_squared_error: 49.323
6 - lr: 5.0000e-04
Epoch 3/10
107/107 [==============================] - 17s 157ms/step - loss: 2221.7703 - mae: 43.3576 - root_mean
_squared_error: 47.1356 - val_loss: 2180.1692 - val_mae: 42.7898 - val_root_mean_squared_error: 46.692
3 - lr: 2.5000e-04
Epoch 4/10
107/107 [==============================] - 17s 157ms/step - loss: 2127.7576 - mae: 42.2777 - root_mean
_squared_error: 46.1276 - val_loss: 2130.1851 - val_mae: 42.2061 - val_root_mean_squared_error: 46.153
9 - lr: 6.2500e-05
Epoch 5/10
107/107 [==============================] - 17s 157ms/step - loss: 2097.5300 - mae: 41.9328 - root_mean
_squared_error: 45.7988 - val_loss: 2118.2615 - val_mae: 42.0580 - val_root_mean_squared_error: 46.024
6 - lr: 1.5625e-05
Epoch 6/10
107/107 [==============================] - 17s 157ms/step - loss: 2094.8672 - mae: 41.8993 - root_mean
_squared_error: 45.7697 - val_loss: 2114.8105 - val_mae: 42.0081 - val_root_mean_squared_error: 45.987
1 - lr: 1.9531e-06
Epoch 7/10
107/107 [==============================] - 17s 158ms/step - loss: 2093.9683 - mae: 41.8894 - root_mean
_squared_error: 45.7599 - val_loss: 2113.5999 - val_mae: 41.9868 - val_root_mean_squared_error: 45.973
9 - lr: 2.4414e-07
Epoch 8/10
107/107 [==============================] - 17s 156ms/step - loss: 2082.9712 - mae: 41.7574 - root_mean
_squared_error: 45.6396 - val_loss: 2113.6479 - val_mae: 41.9813 - val_root_mean_squared_error: 45.974
4 - lr: 1.5259e-08
Epoch 9/10
107/107 [==============================] - 17s 159ms/step - loss: 2090.3999 - mae: 41.8440 - root_mean
_squared_error: 45.7209 - val_loss: 2114.1765 - val_mae: 41.9823 - val_root_mean_squared_error: 45.980
2 - lr: 9.5367e-10
Epoch 10/10
107/107 [==============================] - 17s 160ms/step - loss: 2090.4365 - mae: 41.8466 - root_mean
_squared_error: 45.7213 - val_loss: 2114.4797 - val_mae: 41.9827 - val_root_mean_squared_error: 45.983
5 - lr: 2.9802e-11
Epoch 1/10
107/107 [==============================] - 18s 163ms/step - loss: 1648.5969 - mae: 34.4713 - root_mean
_squared_error: 40.6029 - val_loss: 441.4150 - val_mae: 17.4152 - val_root_mean_squared_error: 21.0099
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 162ms/step - loss: 64.9215 - mae: 5.8992 - root_mean_sq
uared_error: 8.0574 - val_loss: 28.5810 - val_mae: 3.9767 - val_root_mean_squared_error: 5.3461 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 163ms/step - loss: 41.7694 - mae: 4.8479 - root_mean_sq
uared_error: 6.4629 - val_loss: 18.8753 - val_mae: 3.2627 - val_root_mean_squared_error: 4.3446 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 163ms/step - loss: 39.1274 - mae: 4.6824 - root_mean_sq
uared_error: 6.2552 - val_loss: 18.1718 - val_mae: 3.2003 - val_root_mean_squared_error: 4.2628 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 163ms/step - loss: 42.0469 - mae: 4.8539 - root_mean_sq
uared_error: 6.4844 - val_loss: 17.8726 - val_mae: 3.1732 - val_root_mean_squared_error: 4.2276 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 163ms/step - loss: 37.6103 - mae: 4.5827 - root_mean_sq
uared_error: 6.1327 - val_loss: 17.8549 - val_mae: 3.1710 - val_root_mean_squared_error: 4.2255 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 162ms/step - loss: 38.2087 - mae: 4.6698 - root_mean_sq
uared_error: 6.1813 - val_loss: 17.8589 - val_mae: 3.1715 - val_root_mean_squared_error: 4.2260 - lr:
1.2207e-06
```

```
Epoch 8/10
107/107 [==============================] - 17s 162ms/step - loss: 38.7502 - mae: 4.6982 - root_mean_sq
uared_error: 6.2250 - val_loss: 17.8650 - val_mae: 3.1724 - val_root_mean_squared_error: 4.2267 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 159ms/step - loss: 37.5286 - mae: 4.6283 - root_mean_sq
uared_error: 6.1261 - val_loss: 17.8686 - val_mae: 3.1731 - val_root_mean_squared_error: 4.2271 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 159ms/step - loss: 36.8505 - mae: 4.5606 - root_mean_sq
uared_error: 6.0705 - val_loss: 17.8719 - val_mae: 3.1736 - val_root_mean_squared_error: 4.2275 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 164ms/step - loss: 944.4855 - mae: 22.0125 - root_mean_
squared_error: 30.7325 - val_loss: 124.9731 - val_mae: 8.7585 - val_root_mean_squared_error: 11.1791 -
lr: 0.0100
Epoch 2/10
107/107 [==============================] - 17s 162ms/step - loss: 40.9495 - mae: 4.7846 - root_mean_sq
uared_error: 6.3992 - val_loss: 22.7053 - val_mae: 3.6193 - val_root_mean_squared_error: 4.7650 - lr:
0.0050
Epoch 3/10
107/107 [==============================] - 17s 163ms/step - loss: 37.1333 - mae: 4.5771 - root_mean_sq
uared_error: 6.0937 - val_loss: 17.9162 - val_mae: 3.1873 - val_root_mean_squared_error: 4.2328 - lr:
0.0025
Epoch 4/10
107/107 [==============================] - 17s 162ms/step - loss: 36.6208 - mae: 4.5190 - root_mean_sq
uared_error: 6.0515 - val_loss: 17.7484 - val_mae: 3.1695 - val_root_mean_squared_error: 4.2129 - lr:
6.2500e-04
Epoch 5/10
107/107 [==============================] - 17s 163ms/step - loss: 35.1121 - mae: 4.4859 - root_mean_sq
uared_error: 5.9255 - val_loss: 18.0935 - val_mae: 3.2027 - val_root_mean_squared_error: 4.2536 - lr:
1.5625e-04
Epoch 6/10
107/107 [==============================] - 17s 164ms/step - loss: 35.8472 - mae: 4.4771 - root_mean_sq
uared_error: 5.9873 - val_loss: 18.1736 - val_mae: 3.2102 - val_root_mean_squared_error: 4.2630 - lr:
1.9531e-05
Epoch 7/10
107/107 [==============================] - 17s 163ms/step - loss: 34.7049 - mae: 4.3807 - root_mean_sq
uared_error: 5.8911 - val_loss: 18.1867 - val_mae: 3.2114 - val_root_mean_squared_error: 4.2646 - lr:
2.4414e-06
Epoch 8/10
107/107 [==============================] - 17s 162ms/step - loss: 32.7815 - mae: 4.3028 - root_mean_sq
uared_error: 5.7255 - val_loss: 18.1921 - val_mae: 3.2120 - val_root_mean_squared_error: 4.2652 - lr:
1.5259e-07
Epoch 9/10
107/107 [==============================] - 17s 162ms/step - loss: 33.7758 - mae: 4.3712 - root_mean_sq
uared_error: 5.8117 - val_loss: 18.1939 - val_mae: 3.2121 - val_root_mean_squared_error: 4.2654 - lr:
9.5367e-09
Epoch 10/10
107/107 [==============================] - 17s 163ms/step - loss: 34.5680 - mae: 4.4456 - root_mean_sq
uared_error: 5.8795 - val_loss: 18.1944 - val_mae: 3.2122 - val_root_mean_squared_error: 4.2655 - lr:
2.9802e-10
```

In [211…

```
hyp_tun_plot(hists3, 'Validation with a learning rate of', var_lr)
```



As evident from the plot above, low and high learning rates yield comparable performance. Opting for the lower learning rate appears to be the safer choice to avoid potential errors caused by the loss function deviating. Following several rounds of parameter optimization, we have identified our ultimate and most effective model. In the final stage, we will investigate whether adjusting the dropout rate to be faster or slower enhances the overall performance of the model.

In [212…

```
var_drp = [0.1, 0.2, 0.3, 0.5, 0.7]
hists4 = []
for drps in var_drp:
    hists4.append(hyp_tun(BATCH_SIZE, layer_sizes = (32, 64, 128), filter_size = (3, 3), pool_size =
```

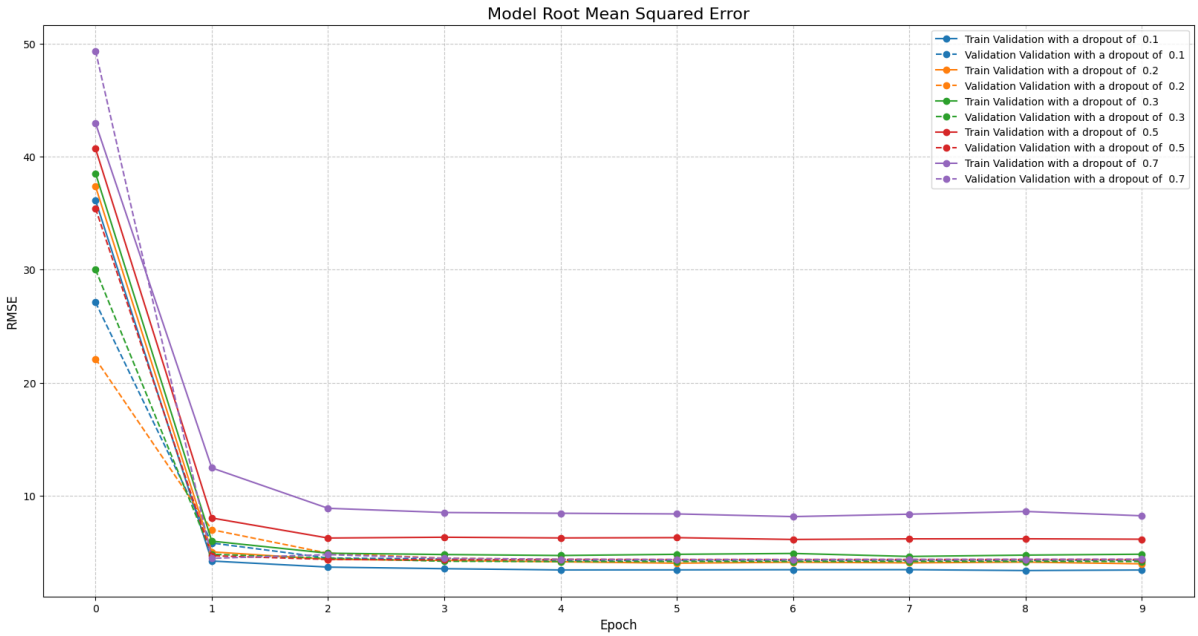
```
Epoch 1/10
107/107 [==============================] - 18s 165ms/step - loss: 1305.9452 - mae: 28.5034 - root_mean
_squared_error: 36.1379 - val_loss: 734.5619 - val_mae: 21.7860 - val_root_mean_squared_error: 27.1028
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 162ms/step - loss: 17.7919 - mae: 3.1289 - root_mean_sq
uared_error: 4.2180 - val_loss: 33.7802 - val_mae: 4.3594 - val_root_mean_squared_error: 5.8121 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 163ms/step - loss: 13.5430 - mae: 2.7717 - root_mean_sq
uared_error: 3.6801 - val_loss: 20.1304 - val_mae: 3.4172 - val_root_mean_squared_error: 4.4867 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 162ms/step - loss: 12.5251 - mae: 2.6962 - root_mean_sq
uared_error: 3.5391 - val_loss: 19.3193 - val_mae: 3.3004 - val_root_mean_squared_error: 4.3954 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 164ms/step - loss: 11.7430 - mae: 2.5810 - root_mean_sq
uared_error: 3.4268 - val_loss: 18.2023 - val_mae: 3.2097 - val_root_mean_squared_error: 4.2664 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 163ms/step - loss: 11.7840 - mae: 2.6012 - root_mean_sq
uared_error: 3.4328 - val_loss: 18.2083 - val_mae: 3.2095 - val_root_mean_squared_error: 4.2671 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 162ms/step - loss: 11.8772 - mae: 2.6099 - root_mean_sq
uared_error: 3.4463 - val_loss: 18.2392 - val_mae: 3.2115 - val_root_mean_squared_error: 4.2707 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 18s 164ms/step - loss: 11.9072 - mae: 2.6117 - root_mean_sq
uared_error: 3.4507 - val_loss: 18.2678 - val_mae: 3.2138 - val_root_mean_squared_error: 4.2741 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 162ms/step - loss: 11.4083 - mae: 2.5307 - root_mean_sq
uared_error: 3.3776 - val_loss: 18.2947 - val_mae: 3.2160 - val_root_mean_squared_error: 4.2772 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 163ms/step - loss: 11.7248 - mae: 2.5819 - root_mean_sq
uared_error: 3.4241 - val_loss: 18.3159 - val_mae: 3.2178 - val_root_mean_squared_error: 4.2797 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 163ms/step - loss: 1396.2247 - mae: 30.1102 - root_mean
_squared_error: 37.3661 - val_loss: 488.5515 - val_mae: 19.8943 - val_root_mean_squared_error: 22.1032
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 159ms/step - loss: 25.2903 - mae: 3.7527 - root_mean_sq
uared_error: 5.0289 - val_loss: 48.8390 - val_mae: 5.1619 - val_root_mean_squared_error: 6.9885 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 157ms/step - loss: 19.0795 - mae: 3.3098 - root_mean_sq
uared_error: 4.3680 - val_loss: 24.0782 - val_mae: 3.6777 - val_root_mean_squared_error: 4.9070 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 157ms/step - loss: 18.1883 - mae: 3.2061 - root_mean_sq
uared_error: 4.2648 - val_loss: 19.9696 - val_mae: 3.3486 - val_root_mean_squared_error: 4.4687 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 157ms/step - loss: 17.2265 - mae: 3.1517 - root_mean_sq
uared_error: 4.1505 - val_loss: 18.9984 - val_mae: 3.2717 - val_root_mean_squared_error: 4.3587 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 157ms/step - loss: 16.2306 - mae: 3.0681 - root_mean_sq
uared_error: 4.0287 - val_loss: 18.8384 - val_mae: 3.2559 - val_root_mean_squared_error: 4.3403 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 158ms/step - loss: 16.9099 - mae: 3.1112 - root_mean_sq
uared_error: 4.1122 - val_loss: 18.7959 - val_mae: 3.2521 - val_root_mean_squared_error: 4.3354 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 17s 159ms/step - loss: 16.4613 - mae: 3.0739 - root_mean_sq
uared_error: 4.0573 - val_loss: 18.7854 - val_mae: 3.2513 - val_root_mean_squared_error: 4.3342 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 157ms/step - loss: 16.9893 - mae: 3.1323 - root_mean_sq
```

```
uared_error: 4.1218 - val_loss: 18.7831 - val_mae: 3.2511 - val_root_mean_squared_error: 4.3339 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 161ms/step - loss: 15.7508 - mae: 3.0201 - root_mean_sq
uared_error: 3.9687 - val_loss: 18.7826 - val_mae: 3.2511 - val_root_mean_squared_error: 4.3339 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 166ms/step - loss: 1483.5975 - mae: 31.6513 - root_mean
_squared_error: 38.5175 - val_loss: 900.1252 - val_mae: 23.2801 - val_root_mean_squared_error: 30.0021
- lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 163ms/step - loss: 35.7181 - mae: 4.4540 - root_mean_sq
uared_error: 5.9765 - val_loss: 23.1871 - val_mae: 3.6904 - val_root_mean_squared_error: 4.8153 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 162ms/step - loss: 24.0963 - mae: 3.7152 - root_mean_sq
uared_error: 4.9088 - val_loss: 19.9253 - val_mae: 3.4050 - val_root_mean_squared_error: 4.4638 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 159ms/step - loss: 22.9621 - mae: 3.6077 - root_mean_sq
uared_error: 4.7919 - val_loss: 17.6156 - val_mae: 3.1755 - val_root_mean_squared_error: 4.1971 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 156ms/step - loss: 22.1557 - mae: 3.5575 - root_mean_sq
uared_error: 4.7070 - val_loss: 17.4176 - val_mae: 3.1405 - val_root_mean_squared_error: 4.1734 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 155ms/step - loss: 23.1437 - mae: 3.6249 - root_mean_sq
uared_error: 4.8108 - val_loss: 17.4367 - val_mae: 3.1416 - val_root_mean_squared_error: 4.1757 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 156ms/step - loss: 23.8974 - mae: 3.6598 - root_mean_sq
uared_error: 4.8885 - val_loss: 17.4446 - val_mae: 3.1424 - val_root_mean_squared_error: 4.1767 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 17s 157ms/step - loss: 21.3687 - mae: 3.5007 - root_mean_sq
uared_error: 4.6226 - val_loss: 17.4476 - val_mae: 3.1427 - val_root_mean_squared_error: 4.1770 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 157ms/step - loss: 22.4908 - mae: 3.5908 - root_mean_sq
uared_error: 4.7425 - val_loss: 17.4486 - val_mae: 3.1428 - val_root_mean_squared_error: 4.1772 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 155ms/step - loss: 23.2182 - mae: 3.6085 - root_mean_sq
uared_error: 4.8185 - val_loss: 17.4490 - val_mae: 3.1429 - val_root_mean_squared_error: 4.1772 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 165ms/step - loss: 1656.9161 - mae: 34.6073 - root_mean
_squared_error: 40.7052 - val_loss: 1254.4232 - val_mae: 30.4428 - val_root_mean_squared_error: 35.417
8 - lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 159ms/step - loss: 64.4059 - mae: 5.8667 - root_mean_sq
uared_error: 8.0253 - val_loss: 21.9620 - val_mae: 3.5430 - val_root_mean_squared_error: 4.6864 - lr:
0.0025
Epoch 3/10
107/107 [==============================] - 17s 160ms/step - loss: 39.0943 - mae: 4.6692 - root_mean_sq
uared_error: 6.2525 - val_loss: 18.9917 - val_mae: 3.2768 - val_root_mean_squared_error: 4.3579 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 17s 162ms/step - loss: 39.9801 - mae: 4.6945 - root_mean_sq
uared_error: 6.3230 - val_loss: 18.6873 - val_mae: 3.2385 - val_root_mean_squared_error: 4.3229 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 157ms/step - loss: 39.1906 - mae: 4.6574 - root_mean_sq
uared_error: 6.2602 - val_loss: 18.9551 - val_mae: 3.2651 - val_root_mean_squared_error: 4.3538 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 155ms/step - loss: 39.5056 - mae: 4.6801 - root_mean_sq
uared_error: 6.2854 - val_loss: 18.9632 - val_mae: 3.2670 - val_root_mean_squared_error: 4.3547 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 158ms/step - loss: 37.5229 - mae: 4.5618 - root_mean_sq
uared_error: 6.1256 - val_loss: 18.9607 - val_mae: 3.2672 - val_root_mean_squared_error: 4.3544 - lr:
1.2207e-06
```

```
Epoch 8/10
107/107 [==============================] - 17s 162ms/step - loss: 38.1909 - mae: 4.6101 - root_mean_sq
uared_error: 6.1799 - val_loss: 18.9611 - val_mae: 3.2674 - val_root_mean_squared_error: 4.3544 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 161ms/step - loss: 38.2573 - mae: 4.6032 - root_mean_sq
uared_error: 6.1852 - val_loss: 18.9612 - val_mae: 3.2675 - val_root_mean_squared_error: 4.3544 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 161ms/step - loss: 37.8005 - mae: 4.6236 - root_mean_sq
uared_error: 6.1482 - val_loss: 18.9613 - val_mae: 3.2675 - val_root_mean_squared_error: 4.3545 - lr:
1.4901e-10
Epoch 1/10
107/107 [==============================] - 18s 165ms/step - loss: 1845.4253 - mae: 37.5646 - root_mean
_squared_error: 42.9584 - val_loss: 2435.6501 - val_mae: 41.3532 - val_root_mean_squared_error: 49.352
3 - lr: 0.0050
Epoch 2/10
107/107 [==============================] - 17s 162ms/step - loss: 155.1619 - mae: 8.8192 - root_mean_s
quared_error: 12.4564 - val_loss: 19.8454 - val_mae: 3.4375 - val_root_mean_squared_error: 4.4548 - l
r: 0.0025
Epoch 3/10
107/107 [==============================] - 17s 163ms/step - loss: 78.9050 - mae: 6.6298 - root_mean_sq
uared_error: 8.8829 - val_loss: 22.8962 - val_mae: 3.6521 - val_root_mean_squared_error: 4.7850 - lr:
0.0012
Epoch 4/10
107/107 [==============================] - 18s 164ms/step - loss: 72.4472 - mae: 6.3460 - root_mean_sq
uared_error: 8.5116 - val_loss: 19.8878 - val_mae: 3.3912 - val_root_mean_squared_error: 4.4596 - lr:
3.1250e-04
Epoch 5/10
107/107 [==============================] - 17s 161ms/step - loss: 71.3014 - mae: 6.3214 - root_mean_sq
uared_error: 8.4440 - val_loss: 19.1369 - val_mae: 3.3153 - val_root_mean_squared_error: 4.3746 - lr:
7.8125e-05
Epoch 6/10
107/107 [==============================] - 17s 158ms/step - loss: 70.3318 - mae: 6.2162 - root_mean_sq
uared_error: 8.3864 - val_loss: 18.9043 - val_mae: 3.2902 - val_root_mean_squared_error: 4.3479 - lr:
9.7656e-06
Epoch 7/10
107/107 [==============================] - 17s 157ms/step - loss: 66.3806 - mae: 6.1051 - root_mean_sq
uared_error: 8.1474 - val_loss: 18.9238 - val_mae: 3.2887 - val_root_mean_squared_error: 4.3502 - lr:
1.2207e-06
Epoch 8/10
107/107 [==============================] - 17s 160ms/step - loss: 69.9474 - mae: 6.2624 - root_mean_sq
uared_error: 8.3635 - val_loss: 19.0065 - val_mae: 3.2936 - val_root_mean_squared_error: 4.3596 - lr:
7.6294e-08
Epoch 9/10
107/107 [==============================] - 17s 158ms/step - loss: 74.0233 - mae: 6.4505 - root_mean_sq
uared_error: 8.6037 - val_loss: 19.0821 - val_mae: 3.2988 - val_root_mean_squared_error: 4.3683 - lr:
4.7684e-09
Epoch 10/10
107/107 [==============================] - 17s 161ms/step - loss: 67.6067 - mae: 6.1459 - root_mean_sq
uared_error: 8.2223 - val_loss: 19.1683 - val_mae: 3.3057 - val_root_mean_squared_error: 4.3782 - lr:
1.4901e-10
```

In [213…

```
hyp_tun_plot(hists4, 'Validation with a dropout of ', var_drp)
```

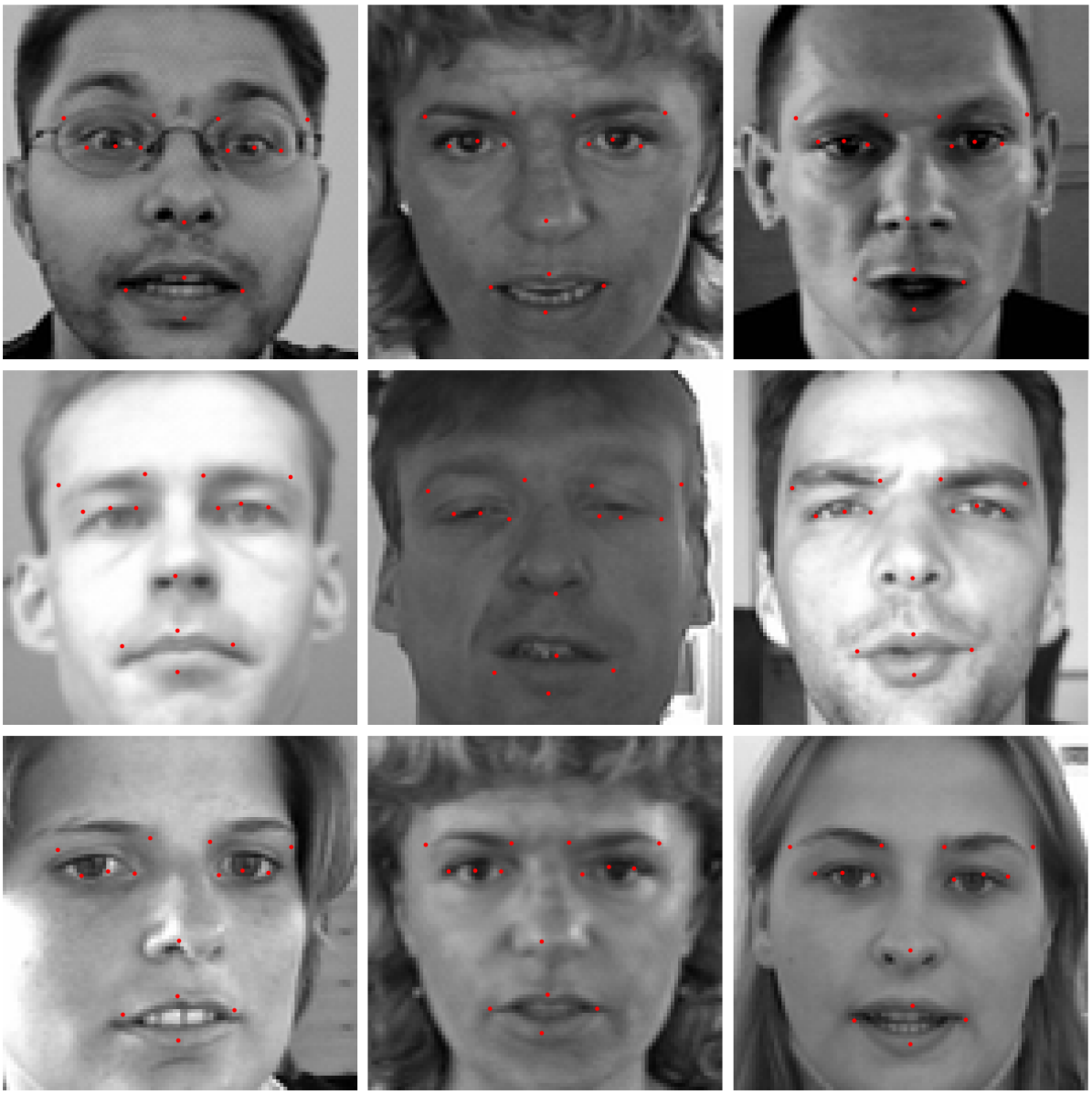


In the concluding phase, we will conduct test predictions using this model and visualize the results in the subsequent plots.

In [214…

```
# Set up a 3x3 grid of subplots
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

for i, ax in enumerate(axes.flatten()):
    # Extract a single test image and its prediction
    test_image = list(dataset_test.skip(i).take(1).as_numpy_iterator())[0][0]
    test_prediction = model.predict(dataset_test.skip(i).take(1))[0]

    # Display the test image
    ax.imshow(test_image, cmap='gray', vmin=0, vmax=255)

    # Scatter plot for predictions
    ax.scatter(test_prediction[0::2], test_prediction[1::2], s=10, c='red', marker='o')

    # Turn off axis labels
    ax.axis("off")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

```
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 37ms/step
```



5. Results and Analysis

Hyperparameter Optimization Summary

Convolution Layer Sizes	Hidden Dense Layer Sizes	Learning Rate	Dropout value	Validation RMSE
64,128,256	256	0.005	0.5	4.382
16,32,64	256	0.005	0.5	5.953
8,16,32	256	0.005	0.5	4.326
32,64,128	128	0.005	0.5	4.5470
32,64,128	64	0.005	0.5	9.8412
32,64,128	32	0.005	0.5	20.3117
32,64,128	16	0.005	0.5	32.4588
32,64,128	256	0.0001	0.5	51.5221
32,64,128	256	0.0005	0.5	50.6472

Convolution Layer Sizes	Hidden Dense Layer Sizes	Learning Rate	Dropout value	Validation RMSE
32,64,128	256	0.001	0.5	45.9835
32,64,128	256	0.01	0.5	4.2655
32,64,128	256	0.005	0.1	4.2797
32,64,128	256	0.005	0.2	4.3339
32,64,128	256	0.005	0.3	4.1772
32,64,128	256	0.005	0.7	4.3782
32,64,128	**256**	**0.005**	**0.5**	**4.2275**

6. Conclusion

We've obtained a model proficient in predicting the positions of facial key features, as evidenced by the example predictions provided. However, even within these limited examples, weaknesses emerge. Upon closer inspection of the middle image, it's evident that the prediction quality is lacking. Additionally, the subject in the image is not facing the camera but rather looking downward. These nuances, compounded by the overall darkness of the image, contribute to suboptimal prediction results. In practical applications, the algorithm would need significant improvement to be viable. In future endeavors, I intend to enhance hyperparameter optimization and explore alternative model architectures. Moreover, I'm intrigued by the prospect of deviating from the constraints of standardized 96x96 grayscale images to develop an algorithm capable of handling arbitrary images featuring faces.

In [ ]: