In [ ]:
```
# I have a tendency to import everthing on the planet so I hope u have ram

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Concatenate, Dense, Flatten, Dropout, Ba
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array,
from tensorflow.keras.applications import ResNet50
from keras.losses import MeanSquaredError

import keras_tuner as kt

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import os

from sklearn.utils.class_weight import compute_class_weight

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
```

Load our data into a pandas df either from csv or excel

In [ ]:
```
df = pd.read_csv('Path1/Path1-Model Training/Path1 Challenge Training Data.csv')

directory = 'Path1/Path1-Model Training/Path1 Challenge Training Images/'
```

In [ ]:
```
# outdated data
# df = pd.read_excel('Path1/Path1-Model Training/Path1 Challenge Training Data.xlsx
# df = df[df['Grade Category'] ! =  'Standard']

# directory = 'Path1/Path1-Model Training/Path1 Challenge Training Images/'
```

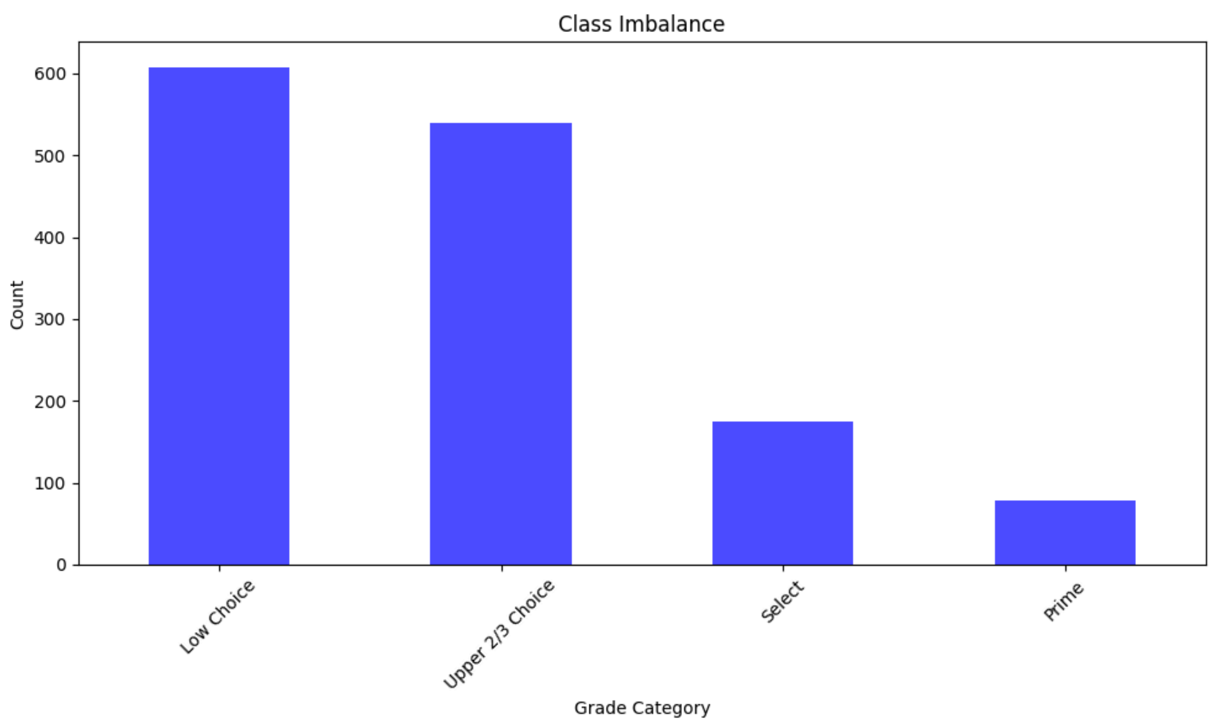
In [ ]:
```
df.head()
```

Out[ ]:

	Filename	Carcass_ID	Score	Grade Category
0	00000002-1.tif	2	526.7	Select
1	00000003-1.tif	3	320.0	Select
2	00000004-1.tif	4	453.3	Select
3	00000005-1.tif	5	566.7	Select
4	00000006-1.tif	6	406.7	Select

In [ ]:

```
# lets check our balance
class_counts = df['Grade Category'].value_counts()
#print(class_counts)

plt.figure(figsize = (10, 6))
class_counts.plot(kind = 'bar', color = 'blue', alpha  =  0.7)
plt.xlabel('Grade Category')
plt.ylabel('Count')
plt.title('Class Imbalance')
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()
```



UH OH!

This stuff is so imbalanced.. what should we do?

Class weights: incentive/bonuses to network to focus on the smaller classes

Undersampling: Reduce samples from larger classes to even the playing field

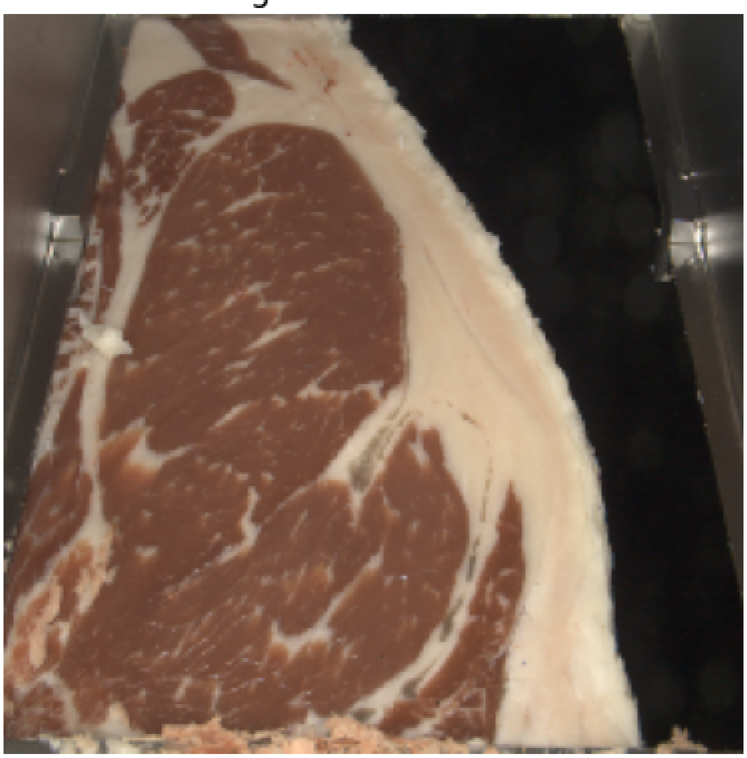
Oversampling: Create artificial samples of underepresented classes. Synthetic Minority Over-sampling Technique or (SMOTE) can be done with either images or conventional data. This technique would generate new data values that exist within the range of existing classes.

In our data today I try our luck at avoiding imabalanced data issues, by relying on the neural networks ability to properly detect the patterns regardless of how much they are represented.

This isn't as dumb as it sounds, because we can implement some regularization techniques like dropout layers.

In [ ]:
```
ith_image = 547

image_filename = df['Filename'].iloc[ith_image]

image_path = os.path.join(directory, image_filename)

# plot
img = image.load_img(image_path, target_size = (224, 224))
plt.imshow(img)
plt.axis('off')
plt.title(f"Image: {image_filename}")
plt.show()

print(df['Grade Category'].iloc[ith_image])
```

Image: 00007277-2.tif



```
Low Choice
```

Split the data frame up

In [ ]:
```
train_data, validation_data = train_test_split(df, test_size = 0.2, random_state =
```

These are data generators

They will convert our dataframe data into images, and then the images into the tensors for the neural network

In [ ]:
```
datagen = ImageDataGenerator(
    rescale = 1./255 #normalization
    # this is where augmentation would occur. we found this to be harmful to our mo
)

batch_size = 32 # do you guys remember batch size?
# it is the amount of data points utilized during a training epoch

train_generator = datagen.flow_from_dataframe(
    dataframe = train_data,
    directory = directory,
    x_col = 'Filename',
    y_col = 'Grade Category',
    target_size = (224, 224),
    color_mode = 'rgb',
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = True
)

validation_generator = datagen.flow_from_dataframe(
    dataframe = validation_data,
    directory = directory,
    x_col = 'Filename',
    y_col = 'Grade Category',
    target_size = (224, 224),
    color_mode = 'rgb',
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = False
)
```

```
Found 1120 validated image filenames belonging to 4 classes.
Found 281 validated image filenames belonging to 4 classes.
```

In [ ]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (224, 224, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation = 'relu'))
```

```
model.add(Flatten())
model.add(Dropout(0.5)) # this dropout layer randomly sets some inputs to 0 within
# this helps prevent overfitting by forcing the neural network to look at new areas

model.add(Dense(256, activation = 'relu') )
model.add(Dense(4, activation = 'softmax'))
```

```
C:\Users\coenp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfr
a8p0\LocalCache\local-packages\Python310\site-packages\keras\src\layers\convolutiona
l\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(
```

What is a convolutional layer?

These are the layers where each node corresponds to a pattern. The first patterns are very simple, and through each layer they combine

What is the 256 node dense layer for?

This layer is the final fully connected layer that does not create any new patterns, but simply looks at all the patterns present, and what each of their presence means for classfication.

What would need to change to create this into a regression problem?

In [ ]:
```
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_9 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_14 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_10 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_15 (Conv2D)	(None, 52, 52, 128)	73,856
flatten_4 (Flatten)	(None, 346112)	0
dropout_5 (Dropout)	(None, 346112)	0
dense_7 (Dense)	(None, 256)	88,604,928
dense_8 (Dense)	(None, 4)	1,028

Total params: 88,699,204 (338.36 MB)

```
Trainable params: 88,699,204 (338.36 MB)
Non-trainable params: 0 (0.00 B)
```

In [ ]:
```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['ac
checkpoint = ModelCheckpoint("coen.keras", monitor = "val_loss", verbose = 1, save_
```

Why did we use categorical crossentropy?

That is because each class is equally distant from eachother. Lets our multi-classifier behave more similarly to a binary classifier

In [ ]:
```
# TTRAINNNN
history = model.fit(
    train_generator,
    steps_per_epoch = batch_size,
    epochs = 8,
    validation_data = validation_generator,
    validation_steps = batch_size,
    callbacks = [checkpoint]
)

# eval on validation
loss, accuracy = model.evaluate(validation_generator, verbose = 1)
print(f"Validation Loss: {loss}")
print(f"Validation Accuracy: {accuracy}")
```

```
Epoch 1/8
C:\Users\coenp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfr
a8p0\LocalCache\local-packages\Python310\site-packages\keras\src\trainers\data_adapt
ers\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class should call `supe
r().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_m
ultiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they w
ill be ignored.
  self._warn_if_super_not_called()
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 990ms/step - accuracy: 0.4235 - loss: 4.3197
Epoch 1: val_loss improved from inf to 0.38808, saving model to coen.keras
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.3056.0_x64__q
bz5n2kfra8p0\lib\contextlib.py:153: UserWarning: Your input ran out of data; interru
pting training. Make sure that your dataset or generator can generate at least `step
s_per_epoch * epochs` batches. You may need to use the `.repeat()` function when bui
lding your dataset.
  self.gen.throw(typ, value, traceback)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 50s 1s/step - accuracy: 0.4272 - loss: 4.2543 - val_accur
acy: 0.8648 - val_loss: 0.3881
Epoch 2/8
Epoch 2/8
 3/32 ━━━━━━━━━━━━━━━━━━━━ 27s 942ms/step - accuracy: 0.8056 - loss: 0.4353
Epoch 2: val_loss improved from 0.38808 to 0.33876, saving model to coen.keras
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 392ms/step - accuracy: 0.8496 - loss: 0.3149 - val_ac
curacy: 0.8790 - val_loss: 0.3388
Epoch 3/8
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 933ms/step - accuracy: 0.8856 - loss: 0.2969
Epoch 3: val_loss improved from 0.33876 to 0.17619, saving model to coen.keras
32/32 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.8857 - loss: 0.2959 - val_accur
acy: 0.9359 - val_loss: 0.1762
Epoch 4/8
 3/32 ━━━━━━━━━━━━━━━━━━━━ 24s 859ms/step - accuracy: 0.9253 - loss: 0.1618
Epoch 4: val_loss improved from 0.17619 to 0.15297, saving model to coen.keras
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 383ms/step - accuracy: 0.9175 - loss: 0.1377 - val_ac
curacy: 0.9217 - val_loss: 0.1530
Epoch 5/8
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 910ms/step - accuracy: 0.9515 - loss: 0.1215
Epoch 5: val_loss improved from 0.15297 to 0.12975, saving model to coen.keras
32/32 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.9517 - loss: 0.1210 - val_accur
acy: 0.9431 - val_loss: 0.1297
Epoch 6/8
 3/32 ━━━━━━━━━━━━━━━━━━━━ 25s 875ms/step - accuracy: 0.9549 - loss: 0.1046
Epoch 6: val_loss did not improve from 0.12975
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 166ms/step - accuracy: 0.9580 - loss: 0.1092 - val_acc
uracy: 0.9324 - val_loss: 0.1311
Epoch 7/8
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 950ms/step - accuracy: 0.9864 - loss: 0.0468
Epoch 7: val_loss improved from 0.12975 to 0.11028, saving model to coen.keras
32/32 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.9865 - loss: 0.0465 - val_accur
acy: 0.9466 - val_loss: 0.1103
Epoch 8/8
 3/32 ━━━━━━━━━━━━━━━━━━━━ 28s 979ms/step - accuracy: 0.9913 - loss: 0.0322
Epoch 8: val_loss did not improve from 0.11028
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 178ms/step - accuracy: 0.9897 - loss: 0.0195 - val_acc
uracy: 0.9431 - val_loss: 0.1124
9/9 ━━━━━━━━━━━━━━━━━━━━ 4s 377ms/step - accuracy: 0.9542 - loss: 0.0998
Validation Loss: 0.12488292157649994
Validation Accuracy: 0.9430605173110962
```

In [ ]:

```
model_path = 'C:/Users/coenp/Documents/CS345/usda/coen.keras'
model = load_model(model_path)
```

In [ ]:

```
class_names = list(validation_generator.class_indices.keys())
class_indices = list(validation_generator.class_indices.values())

for idx, name in zip(class_indices, class_names):
    print(f"Class index {idx} corresponds to {name}")
```

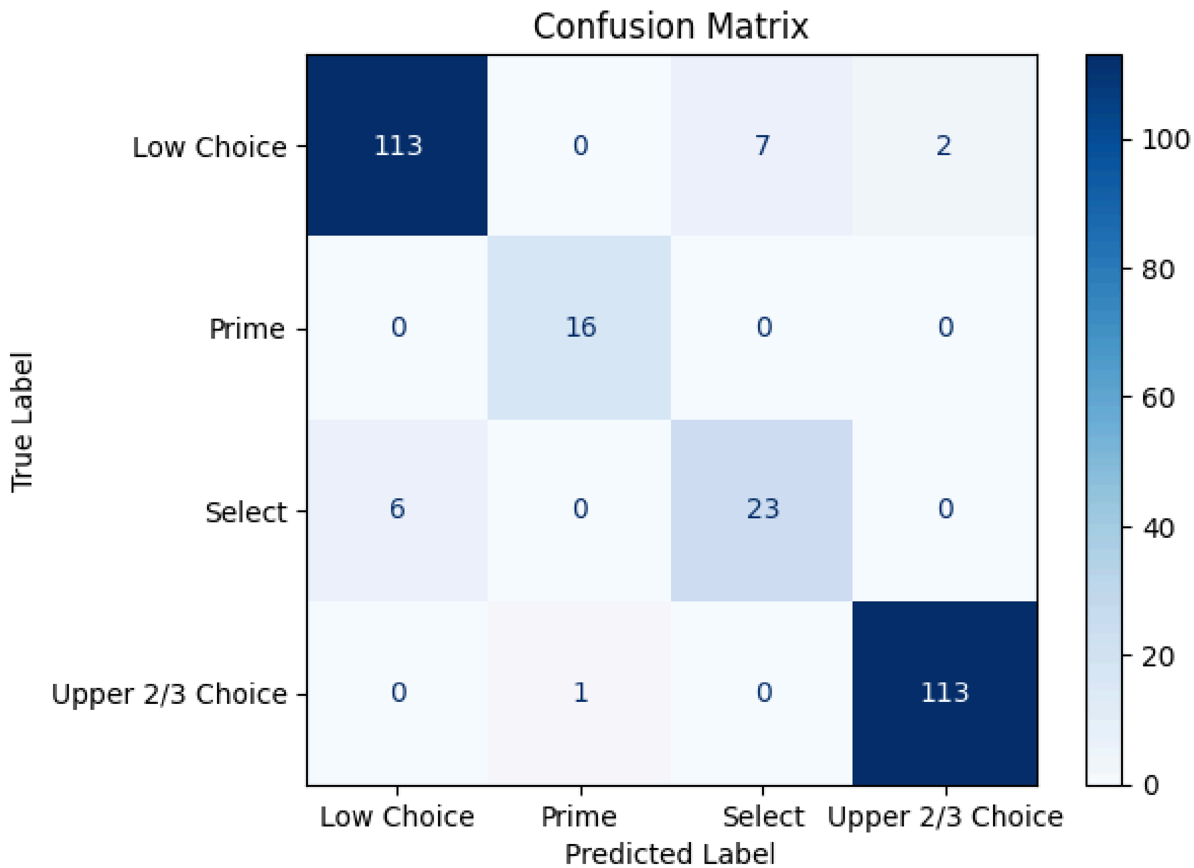
```
Class index 0 corresponds to Low Choice
Class index 1 corresponds to Prime
Class index 2 corresponds to Select
Class index 3 corresponds to Upper 2/3 Choice
```

In [ ]:
```
y_pred = model.predict(validation_generator)
predicted_classes = np.argmax(y_pred, axis = 1)
print(predicted_classes)

true_labels = validation_generator.classes
# print(true_labels)
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 370ms/step
[3 3 2 3 0 0 3 3 0 0 2 0 0 0 2 0 0 3 0 3 0 2 0 3 3 0 0 0 3 3 3 2 3 3 3 3 3
 0 1 1 3 0 3 1 3 3 3 0 0 2 2 3 0 2 2 0 3 3 0 3 3 3 0 0 3 3 2 0 1 0 0 0 0 0
 0 3 3 3 1 2 3 0 0 3 0 0 0 0 3 3 0 3 2 3 0 0 3 0 0 0 2 0 0 0 3 3 0 3 0 3 3
 3 0 2 2 0 1 2 3 3 0 0 0 2 0 3 3 0 0 0 0 3 3 3 3 2 3 3 0 0 0 0 3 3 0 0 0 3
 3 0 0 0 3 0 0 3 0 0 3 0 1 3 3 0 3 0 3 2 0 2 3 3 3 2 2 0 0 0 3 0 3 3 0 0 3
 3 0 0 3 0 3 3 2 1 3 3 0 0 2 3 0 3 0 1 3 0 0 2 3 3 0 3 3 3 3 1 3 1 3 0 0 1
 3 3 3 0 0 0 3 0 3 3 0 3 0 0 3 1 0 3 0 0 3 2 1 0 0 0 3 0 1 3 3 3 0 2 0 0 0
 2 3 3 0 0 1 3 3 3 1 2 0 3 3 0 3 0 2 0 3 3 0]
```

Let's look at our confusion matrix

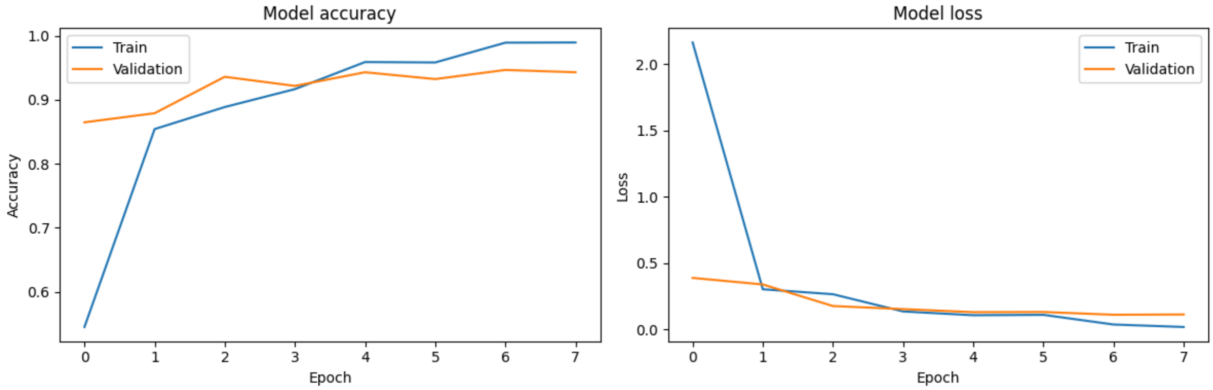
In [ ]:
```
cm = confusion_matrix(true_labels, predicted_classes)

# Plot confusion matrix
labels = list(validation_generator.class_indices.keys())
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = labels)
disp.plot(cmap = plt.cm.Blues)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Plot the accuracy and loss

Could probably benefit from more epochs but i value my computer's life

In [ ]:
```
plt.figure(figsize = (12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')

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

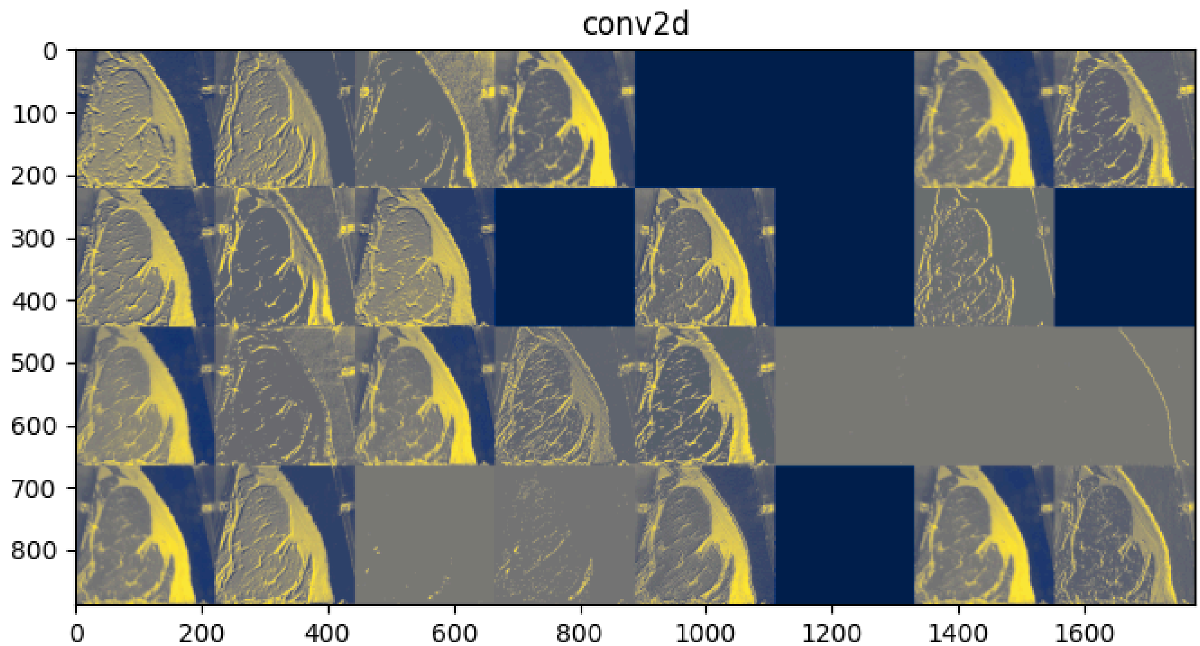


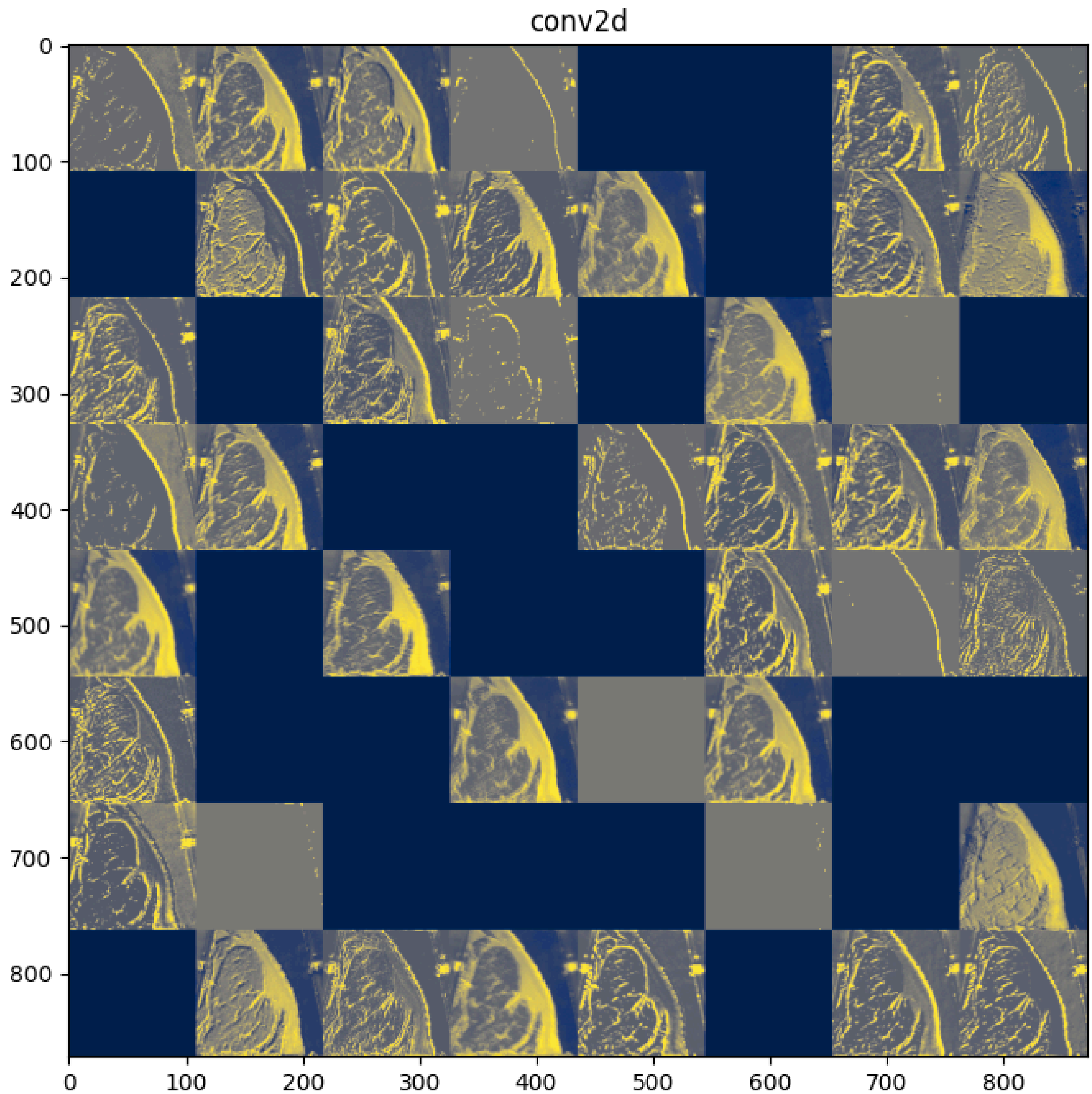
Heatmap time

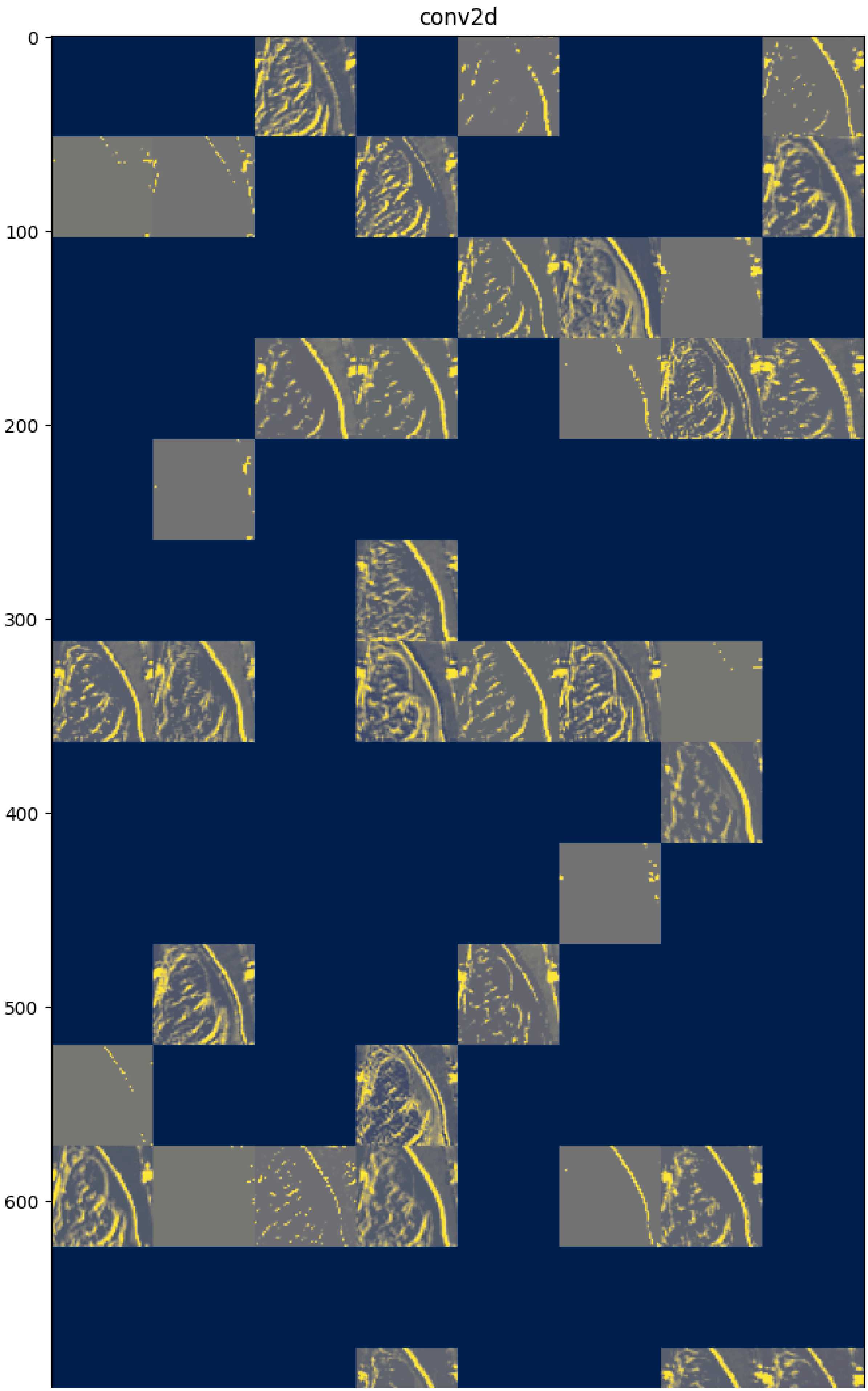
In [ ]:
```
img_array = img_to_array(img)
img_array /=  255.0

# this model holds the outputs for our original models layers
activation_model = Model(inputs = model.layers[0].input, outputs = [layer.output fo

# pulls the activations for the inputted image from each layer in our new model
activations = activation_model.predict(img_array.reshape(1, 224, 224, 3)) # we turn

# Plot the activations for each convolutional layer
for current_layer in activations:

    # pulls the size of the current layer to format grid
    n_features = current_layer.shape[-1]
    size = current_layer.shape[1]
    n_cols = n_features // 8
    display_grid = np.zeros((size * n_cols, 8 * size))

    for col in range(n_cols):
        for row in range(8):

            #extracts the current pattern in the layer
            channel_image = current_layer[0, :, :, col * 8 + row]

            #standardize and reshift images back to a 0-255 scale
            channel_image -=  channel_image.mean()
            channel_image /=  channel_image.std()
            channel_image *=  64
            channel_image +=  128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')

            #this adds each nodes image/pattern to the current grid
            display_grid[col * size : (col + 1) * size, row * size : (row + 1) * si

    scale = 1. / size
    plt.figure(figsize = (scale * display_grid.shape[1], scale * display_grid.shape
    plt.title("conv2d")
    plt.grid(False)
    plt.imshow(display_grid, aspect = 'auto', cmap = 'cividis')
    #other colors inferno, plasma, magma, bone
```

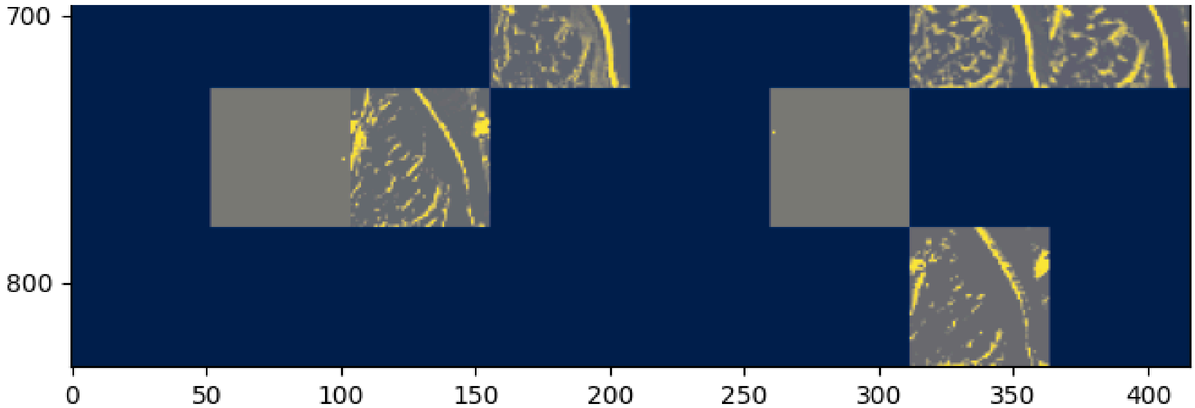
```
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
C:\Users\coenp\AppData\Local\Temp\ipykernel_9884\3946989386.py:27: RuntimeWarning: i
nvalid value encountered in divide
  channel_image /=  channel_image.std()
C:\Users\coenp\AppData\Local\Temp\ipykernel_9884\3946989386.py:30: RuntimeWarning: i
nvalid value encountered in cast
  channel_image = np.clip(channel_image, 0, 255).astype('uint8')
```




conv2d
0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
800


conv2d
0
100
200
300
400
500
600



How could this model be better?

One technique we spent a lot of time looking into was

transfer learning.

Trasnfer learning is the use of implementing a pre-trained model from another task and repurposing it for new data. How could this be possible and where do I find it?

Resnet is the name of another model that has already been trained on millions of different photos. What would the benefit of this be?

The basic idea is that images can have the same basic patterns as others, and implementing a Resnet model in the beginning of your own would allow you to skip that basic pattern detection, and only focus on the larget patterns a CNN may need to detect for your new data.

Here's how you would setup code for incorporating a base model

In [ ]:
```
base_model = ResNet50(weights = 'imagenet', include_top = False, input_shape = (224

# Here we freeze the layers inside of resnet. Meaning that they will not be retrain
for layer in base_model.layers:
    layer.trainable = False
```

In [ ]:
```
# Here we load the basemodel and add layers on top of it
model = Sequential([
    base_model,
    layers.Conv2D(64, (3, 3), activation = 'relu'),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation = 'relu'),
    layers.Dropout(0.5),
    layers.Dense(4, activation = 'softmax')
])

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['ac
```

```
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	?	23,587,712
conv2d_16 (Conv2D)	?	0 (unbuilt)
max_pooling2d_11 (MaxPooling2D)	?	0 (unbuilt)
global_average_pooling2d_1 (GlobalAveragePooling2D)	?	0 (unbuilt)
dense_9 (Dense)	?	0 (unbuilt)
dropout_6 (Dropout)	?	0
dense_10 (Dense)	?	0 (unbuilt)

Total params: 23,587,712 (89.98 MB)

Trainable params: 0 (0.00 B)

Non-trainable params: 23,587,712 (89.98 MB)

Let's do regression on score

In [ ]:

```
train_generator_score = datagen.flow_from_dataframe(
    dataframe = train_data,
    directory = directory,
    x_col = 'Filename',
    y_col = 'Score',
    target_size = (224, 224),
    color_mode = 'rgb',
    batch_size = batch_size,
    class_mode = 'raw',
    shuffle = True
)

validation_generator_score = datagen.flow_from_dataframe(
    dataframe = validation_data,
    directory = directory,
    x_col = 'Filename',
    y_col = 'Score',
    target_size = (224, 224),
    color_mode = 'rgb',
    batch_size = batch_size,
    class_mode = 'raw',
    shuffle = False
)
```

```
Found 1120 validated image filenames.
Found 281 validated image filenames.
```

In [ ]:
```
model_score = Sequential()

model_score.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (224, 224, 3)
model_score.add(MaxPooling2D((2, 2)))

model_score.add(Conv2D(64, (3, 3), activation = 'relu'))
model_score.add(MaxPooling2D((2, 2)))

model_score.add(Conv2D(128, (3, 3), activation = 'relu'))

model_score.add(Flatten())
model_score.add(Dropout(0.5))

model_score.add(Dense(256, activation = 'relu') )
model_score.add(Dense(1, activation = 'linear', name = 'Score'))
# change output layer to one, with linear activation
```

```
C:\Users\coenp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfr
a8p0\LocalCache\local-packages\Python310\site-packages\keras\src\layers\convolutiona
l\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(
```

In [ ]:
```
model_score.compile(optimizer = 'Adam', loss = MeanSquaredError(), metrics = ['mae'

checkpoint = ModelCheckpoint("coen_regression.keras", monitor = 'val_loss', verbose
)
```

In [ ]:
```
# TTRAINNNN
score_history = model_score.fit(
    train_generator_score,
    epochs = 8,
    validation_data = validation_generator_score,
    callbacks = [checkpoint]
)
```

```
Epoch 1/8
```

```
C:\Users\coenp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfr
a8p0\LocalCache\local-packages\Python310\site-packages\keras\src\trainers\data_adapt
ers\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class should call `supe
r().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_m
ultiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they w
ill be ignored.
  self._warn_if_super_not_called()
```

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 923ms/step - loss: 75057.7266 - mae: 209.5457
Epoch 1: val_loss improved from inf to 12549.88672, saving model to coen_regression.
keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 50s 1s/step - loss: 73917.4219 - mae: 207.4794 - val_los
s: 12549.8867 - val_mae: 91.0044
Epoch 2/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 12525.1162 - mae: 88.2510
Epoch 2: val_loss improved from 12549.88672 to 10079.94922, saving model to coen_reg
ression.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - loss: 12521.9971 - mae: 88.2151 - val_loss:
10079.9492 - val_mae: 79.7376
Epoch 3/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 11154.6025 - mae: 82.4667
Epoch 3: val_loss improved from 10079.94922 to 9718.77539, saving model to coen_regr
ession.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - loss: 11150.5703 - mae: 82.4453 - val_loss:
9718.7754 - val_mae: 80.8360
Epoch 4/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 10435.8975 - mae: 81.2256
Epoch 4: val_loss improved from 9718.77539 to 9460.17090, saving model to coen_regre
ssion.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - loss: 10460.7607 - mae: 81.3064 - val_loss:
9460.1709 - val_mae: 80.1255
Epoch 5/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 10328.2305 - mae: 80.3212
Epoch 5: val_loss improved from 9460.17090 to 8095.65527, saving model to coen_regre
ssion.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - loss: 10336.4600 - mae: 80.3239 - val_loss:
8095.6553 - val_mae: 71.4298
Epoch 6/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 9491.8018 - mae: 76.3536
Epoch 6: val_loss did not improve from 8095.65527
35/35 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - loss: 9494.9072 - mae: 76.3694 - val_loss:
10408.0078 - val_mae: 86.1160
Epoch 7/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 8884.6367 - mae: 74.6315
Epoch 7: val_loss improved from 8095.65527 to 7798.22412, saving model to coen_regre
ssion.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - loss: 8901.0557 - mae: 74.6943 - val_loss:
7798.2241 - val_mae: 71.4449
Epoch 8/8
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - loss: 8652.3506 - mae: 73.3495
Epoch 8: val_loss improved from 7798.22412 to 7477.49316, saving model to coen_regre
ssion.keras
35/35 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - loss: 8652.0967 - mae: 73.3366 - val_loss:
7477.4932 - val_mae: 68.0459
```

In [ ]:

```python
predicted_scores = model_score.predict(validation_generator_score)

lowest_score = np.min(predicted_scores)
highest_score = np.max(predicted_scores)

print(f"Lowest Score: {lowest_score}")
print(f"Highest Score: {highest_score}")
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 291ms/step
Lowest Score: 371.3567199707031
Highest Score: 568.480712890625
```