```
from google.colab import drive
drive.mount('/content/drive')

base_dir = '/content/drive/My Drive/Senior Year/Applied Data Science'
metadata_path = f'{base_dir}/sample_reduced_features.csv'
image_dir = f'{base_dir}/Data/downsampled_data/'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Dense, Flatten, Concatenate
from tensorflow.keras.models import Model

metadata_path = '/content/drive/My Drive/Senior Year/Applied Data Science/sample_reduced_features.csv'
metadata = pd.read_csv(metadata_path)

image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/downsampled_data/'

metadata['Image_Path'] = metadata['Simplified_FileName'] + '_median_aggregated.tiff'
image_paths = image_dir + metadata['Image_Path']

meta_features = ['area', 'mean_intensity', 'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3', 'eccentricity', 'solidity']
X_meta = metadata[meta_features]
y_labels = metadata['Metadata_pert_iname']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_labels)

scaler = StandardScaler()
X_meta_scaled = scaler.fit_transform(X_meta)

X_meta_train, X_meta_val, y_train, y_val = train_test_split(X_meta_scaled, y_encoded, test_size=0.2, random_state=42)
image_paths_train, image_paths_val = train_test_split(image_paths, test_size=0.2, random_state=42)

def load_image(path):
    img = tf.keras.utils.load_img(path.numpy().decode('utf-8'), target_size=(64, 64))
    img = tf.keras.utils.img_to_array(img) / 255.0
    return img

def hybrid_data_tf_generator(image_paths, X_meta, labels, batch_size=32):
    image_paths = tf.convert_to_tensor(image_paths.values, dtype=tf.string)
    dataset = tf.data.Dataset.from_tensor_slices((image_paths, X_meta, labels))

    def process_data(img_path, meta_data, label):
        img = tf.py_function(func=load_image, inp=[img_path], Tout=tf.float32)
        img.set_shape((64, 64, 3))
        return {"image_input": img, "meta_input": meta_data}, label

    dataset = dataset.map(process_data, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.shuffle(buffer_size=1000).batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

image_input = Input(shape=(64, 64, 3), name="image_input")
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(image_input)
x = Flatten()(x)

meta_input = Input(shape=(X_meta_train.shape[1],), name="meta_input")
meta_features = Dense(32, activation='relu')(meta_input)

combined = Concatenate()([x, meta_features])
output = Dense(len(label_encoder.classes_), activation='softmax')(combined)

hybrid_model = Model(inputs={"image_input": image_input, "meta_input": meta_input}, outputs=output)
hybrid_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

train_dataset = hybrid_data_tf_generator(image_paths_train, X_meta_train, y_train, batch_size=32)
val_dataset = hybrid_data_tf_generator(image_paths_val, X_meta_val, y_val, batch_size=32)

history = hybrid_model.fit(
    train_dataset,
    validation_data=val_dataset
```

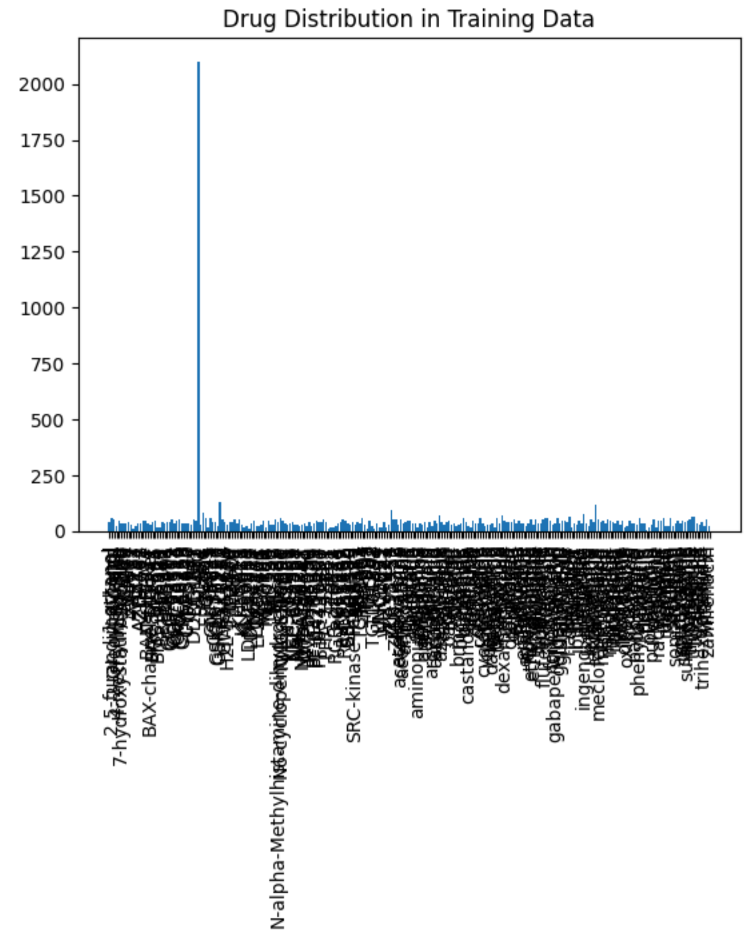
```
    validation_data=val_dataset,
    epochs=30,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=3, min_lr=1e-5)
    ],
    verbose=1
)
```

```
Epoch 1/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 73s 181ms/step - accuracy: 0.4598 - loss: 3.4246 - val_accuracy: 0.9697 - val_loss: 0.2843 - le
Epoch 2/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 56s 143ms/step - accuracy: 0.9963 - loss: 0.0352 - val_accuracy: 0.9752 - val_loss: 0.2651 - le
Epoch 3/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 56s 143ms/step - accuracy: 0.9992 - loss: 0.0118 - val_accuracy: 0.9759 - val_loss: 0.2795 - le
Epoch 4/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 56s 145ms/step - accuracy: 0.9995 - loss: 0.0079 - val_accuracy: 0.9772 - val_loss: 0.2933 - le
Epoch 5/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 57s 145ms/step - accuracy: 0.9997 - loss: 0.0054 - val_accuracy: 0.9772 - val_loss: 0.2950 - le
Epoch 6/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 56s 143ms/step - accuracy: 0.9995 - loss: 0.0078 - val_accuracy: 0.9772 - val_loss: 0.2967 - le
Epoch 7/30
363/363 ━━━━━━━━━━━━━━━━━━━━ 55s 142ms/step - accuracy: 0.9995 - loss: 0.0082 - val_accuracy: 0.9772 - val_loss: 0.2977 - le
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import load_img, img_to_array

example_image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/example_data/'

example_image_paths = [os.path.join(example_image_dir, fname) for fname in os.listdir(example_image_dir) if fname.endswith('.tif

def preprocess_images(paths):
    images = []
    for path in paths:
        img = load_img(path, target_size=(64, 64))
        img_array = img_to_array(img) / 255.0
        images.append(img_array)
    return np.array(images)

X_image_example = preprocess_images(example_image_paths)

dummy_meta = np.zeros((len(X_image_example), X_meta_train.shape[1]))

predictions = hybrid_model.predict({"image_input": X_image_example, "meta_input": dummy_meta})

predicted_drugs = label_encoder.inverse_transform(np.argmax(predictions, axis=1))

for i, path in enumerate(example_image_paths):
    print(f"Image: {os.path.basename(path)}, Predicted Drug: {predicted_drugs[i]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
Image: r04c08f05p01-compound-FK866.tiff, Predicted Drug: CC-401
Image: r04c14f05p01-compound-DMSO.tiff, Predicted Drug: CC-401
Image: r06c10f05p01-compound-quinidine.tiff, Predicted Drug: CC-401
Image: r12c09f05p01-compound-FK866.tiff, Predicted Drug: CC-401
Image: r13c02f05p01-compound-LY2109761.tiff, Predicted Drug: CC-401
```

```
import matplotlib.pyplot as plt
unique, counts = np.unique(y_train, return_counts=True)
drug_counts = dict(zip(label_encoder.inverse_transform(unique), counts))

plt.bar(drug_counts.keys(), drug_counts.values())
plt.xticks(rotation=90)
plt.title("Drug Distribution in Training Data")
plt.show()
```



This indicates bias towards certain drug, possibly CC-401.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.layers import Input, Dense, Flatten, Concatenate, Conv2D
from tensorflow.keras.models import Model

metadata_path = '/content/drive/My Drive/Senior Year/Applied Data Science/sample_reduced_features.csv'
metadata = pd.read_csv(metadata_path)

image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/downsampled_data/'

metadata['Image_Path'] = metadata['Simplified_FileName'] + '_median_aggregated.tiff'
image_paths = image_dir + metadata['Image_Path']

meta_features = ['area', 'mean_intensity', 'bbox-0', 'bbox-1', 'bbox-2', 'bbox-3', 'eccentricity', 'solidity']
X_meta = metadata[meta_features]
y_labels = metadata['Metadata_pert_iname']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_labels)

scaler = StandardScaler()
X_meta_scaled = scaler.fit_transform(X_meta)

X_meta_train, X_meta_val, y_train, y_val = train_test_split(X_meta_scaled, y_encoded, test_size=0.2, random_state=42)
image_paths_train, image_paths_val = train_test_split(image_paths, test_size=0.2, random_state=42)

class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = dict(enumerate(class_weights))

def load_image(path):
    img = tf.keras.utils.load_img(path, target_size=(64, 64))
    return tf.keras.utils.img_to_array(img) / 255.0

def hybrid_data_generator(image_paths, X_meta, labels, class_weights, batch_size=32):
    num_samples = len(labels)
    while True:
        for offset in range(0, num_samples, batch_size):
            batch_paths = image_paths.iloc[offset:offset + batch_size].values
            batch_images = np.array([load_image(path) for path in batch_paths])
            batch_meta = X_meta[offset:offset + batch_size]
            batch_labels = labels[offset:offset + batch_size]
            batch_weights = np.array([class_weights[label] for label in batch_labels])
            yield {"image_input": batch_images, "meta_input": batch_meta}, batch_labels, batch_weights

image_input = Input(shape=(64, 64, 3), name="image_input")
x = Conv2D(32, (3, 3), activation='relu')(image_input)
x = Flatten()(x)

meta_input = Input(shape=(X_meta_train.shape[1],), name="meta_input")
meta_features = Dense(32, activation='relu')(meta_input)

combined = Concatenate()([x, meta_features])
output = Dense(len(label_encoder.classes_), activation='softmax')(combined)

hybrid_model = Model(inputs={"image_input": image_input, "meta_input": meta_input}, outputs=output)
hybrid_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

train_gen = hybrid_data_generator(image_paths_train, X_meta_train, y_train, class_weight_dict, batch_size=32)
val_data = {
    "image_input": np.array([load_image(path) for path in image_paths_val[:32]]),
    "meta_input": X_meta_val[:32]
}

history = hybrid_model.fit(
    train_gen,
    steps_per_epoch=len(y_train) // 32,
    validation_data=(val_data, y_val[:32]),
    epochs=30,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=3, min_lr=1e-5)
    ],
    verbose=1
```

```
)
```

```
Epoch 1/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 182s 496ms/step - accuracy: 0.4018 - loss: 3.6175 - val_accuracy: 0.9375 - val_loss: 0.2972 - l
Epoch 2/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 180s 494ms/step - accuracy: 0.9855 - loss: 0.0831 - val_accuracy: 0.9688 - val_loss: 0.2776 - l
Epoch 3/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 179s 496ms/step - accuracy: 0.9988 - loss: 0.0125 - val_accuracy: 0.9688 - val_loss: 0.3002 - l
Epoch 4/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 179s 495ms/step - accuracy: 0.9998 - loss: 0.0102 - val_accuracy: 0.9688 - val_loss: 0.3027 - l
Epoch 5/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 179s 494ms/step - accuracy: 0.9998 - loss: 0.0099 - val_accuracy: 0.9688 - val_loss: 0.3020 - l
Epoch 6/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 179s 494ms/step - accuracy: 0.9998 - loss: 0.0098 - val_accuracy: 0.9688 - val_loss: 0.3038 - l
Epoch 7/30
362/362 ━━━━━━━━━━━━━━━━━━━━ 181s 500ms/step - accuracy: 0.9998 - loss: 0.0097 - val_accuracy: 0.9688 - val_loss: 0.3055 - l
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import load_img, img_to_array

example_data_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/example_data/'
example_image_paths = [os.path.join(example_data_dir, fname) for fname in os.listdir(example_data_dir) if fname.endswith('.tiff'

def load_example_images(image_paths):
    images = [img_to_array(load_img(path, target_size=(64, 64))) / 255.0 for path in image_paths]
    return np.array(images)

example_images = load_example_images(example_image_paths)

predictions = hybrid_model.predict({"image_input": example_images, "meta_input": np.zeros((len(example_images), X_meta_train.sha

predicted_labels = label_encoder.inverse_transform(np.argmax(predictions, axis=1))

for img_path, pred_label in zip(example_image_paths, predicted_labels):
    print(f"Image: {os.path.basename(img_path)}, Predicted Drug: {pred_label}")
```

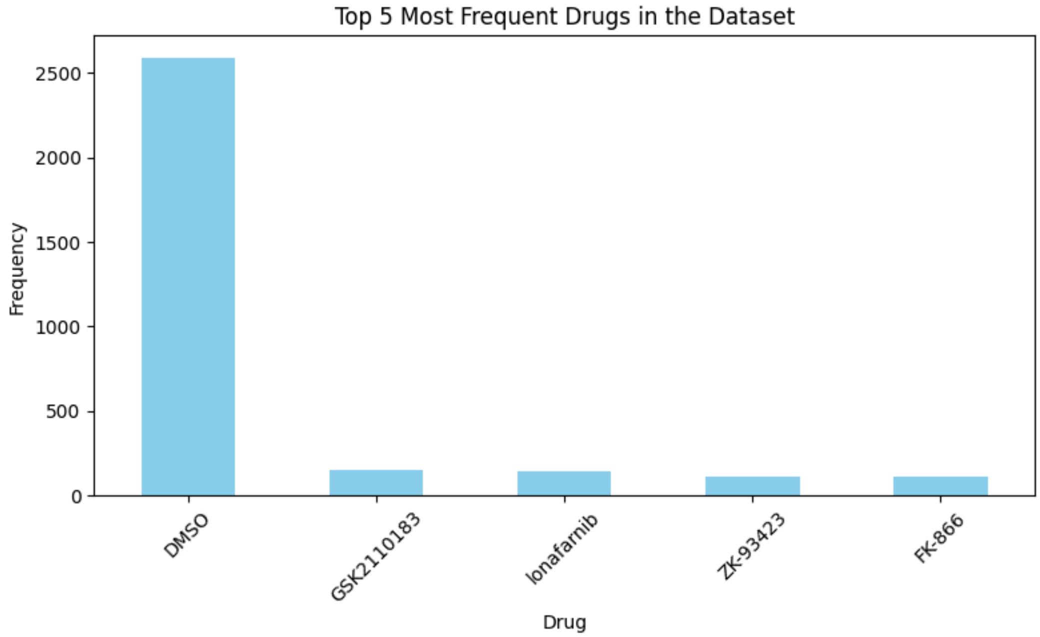
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
Image: r04c08f05p01-compound-FK866.tiff, Predicted Drug: SGX523
Image: r04c14f05p01-compound-DMSO.tiff, Predicted Drug: SGX523
Image: r06c10f05p01-compound-quinidine.tiff, Predicted Drug: SGX523
Image: r12c09f05p01-compound-FK866.tiff, Predicted Drug: SGX523
Image: r13c02f05p01-compound-LY2109761.tiff, Predicted Drug: SGX523
```

```
import pandas as pd
import os
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.models import Model

metadata_path = '/content/drive/My Drive/Senior Year/Applied Data Science/sample_reduced_features.csv'
metadata = pd.read_csv(metadata_path)

image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/downsampled_data/'

metadata['Image_Path'] = image_dir + metadata['Simplified_FileName'] + '_median_aggregated.tiff'
image_paths = metadata['Image_Path']

y_labels = metadata['Metadata_pert_iname']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_labels)

image_paths_train, image_paths_val, y_train, y_val = train_test_split(
    image_paths, y_encoded, test_size=0.2, random_state=42
)

def load_image(path):
    img = tf.keras.utils.load_img(path, target_size=(64, 64))
    return tf.keras.utils.img_to_array(img) / 255.0

def image_data_generator(image_paths, labels, batch_size=32):
    num_samples = len(labels)
    while True:
        for offset in range(0, num_samples, batch_size):
            batch_paths = image_paths.iloc[offset:offset + batch_size].values
            batch_images = np.array([load_image(path) for path in batch_paths])
            batch_labels = labels[offset:offset + batch_size]
            yield batch_images, batch_labels

image_input = Input(shape=(64, 64, 3), name="image_input")
x = Conv2D(16, (3, 3), activation='relu')(image_input)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
output = Dense(len(label_encoder.classes_), activation='softmax')(x)

simplified_model = Model(inputs=image_input, outputs=output)
simplified_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

train_gen = image_data_generator(image_paths_train, y_train, batch_size=32)
val_gen = image_data_generator(image_paths_val, y_val, batch_size=32)

history = simplified_model.fit(
    train_gen,
    steps_per_epoch=len(y_train) // 32,
    validation_data=val_gen,
    validation_steps=len(y_val) // 32,
    epochs=20,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=2, min_lr=1e-5)
    ],
    verbose=1
)
```

```
Epoch 1/20
 41/362 ━━━━━━━━━━━━━━━━━━━━ 2:35 484ms/step - accuracy: 0.1352 - loss: 5.3415
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-26-c6744bbbe8ee> in <cell line: 66>()
     64 
     65 # Train the model
---> 66 history = simplified_model.fit(
     67     train_gen,
     68     steps_per_epoch=len(y_train) // 32,

10 frames

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs,
attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

```
import pandas as pd

metadata_path = '/content/drive/My Drive/Senior Year/Applied Data Science/sample_reduced_features.csv'
metadata = pd.read_csv(metadata_path)

drug_counts = metadata['Metadata_pert_iname'].value_counts()

top_5_drugs = drug_counts.head(5)

print("Top 5 most frequent drugs in the dataset:")
print(top_5_drugs)

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
top_5_drugs.plot(kind='bar', color='skyblue')
plt.title('Top 5 Most Frequent Drugs in the Dataset')
plt.xlabel('Drug')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Top 5 most frequent drugs in the dataset:
Metadata_pert_iname
DMSO          2592
GSK2110183     153
lonafarnib     146
ZK-93423       113
FK-866         109
Name: count, dtype: int64
```




```
!pip install cellpose

from cellpose import models, io
import os
import pandas as pd
import numpy as np
from skimage.measure import regionprops, label
from tqdm import tqdm

\image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/downsampled_data'
output_csv_path = '/content/drive/My Drive/Senior Year/Applied Data Science/cellpose_features.csv'

model = models.Cellpose(gpu=True, model_type='cyto')

features = []

for filename in tqdm(os.listdir(image_dir)):
    if filename.endswith('.tiff'):
        img_path = os.path.join(image_dir, filename)
        img = io.imread(img_path)

        outputs = model.eval(img, channels=[0, 0], diameter=None, flow_threshold=0.4, cellprob_threshold=0.0)
        masks = outputs[0]

        props = regionprops(label(masks), intensity_image=img)
        for prop in props:
            features.append({
                'Image': filename,
                'Area': prop.area,
                'Mean Intensity': prop.mean_intensity,
                'Bounding Box Sum': sum(prop.bbox),
                'Eccentricity': prop.eccentricity,
                'Solidity': prop.solidity
            })

df_features = pd.DataFrame(features)
df_features.to_csv(output_csv_path, index=False)

print(f"Feature extraction complete. Results saved to {output_csv_path}")
```

```
Requirement already satisfied: cellpose in /usr/local/lib/python3.10/dist-packages (3.1.0)
Requirement already satisfied: numpy<2.1,>=1.20.0 in /usr/local/lib/python3.10/dist-packages (from cellpose) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from cellpose) (1.13.1)
Requirement already satisfied: natsort in /usr/local/lib/python3.10/dist-packages (from cellpose) (8.4.0)
Requirement already satisfied: tifffile in /usr/local/lib/python3.10/dist-packages (from cellpose) (2024.9.20)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from cellpose) (4.66.6)
Requirement already satisfied: numba>=0.53.0 in /usr/local/lib/python3.10/dist-packages (from cellpose) (0.60.0)
Requirement already satisfied: llvmlite in /usr/local/lib/python3.10/dist-packages (from cellpose) (0.43.0)
Requirement already satisfied: torch>=1.6 in /usr/local/lib/python3.10/dist-packages (from cellpose) (2.5.0+cu121)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.10/dist-packages (from cellpose) (4.10.0.84)
Requirement already satisfied: fastremap in /usr/local/lib/python3.10/dist-packages (from cellpose) (1.15.0)
Requirement already satisfied: imagecodecs in /usr/local/lib/python3.10/dist-packages (from cellpose) (2024.9.22)
Requirement already satisfied: roifile in /usr/local/lib/python3.10/dist-packages (from cellpose) (2024.9.15)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpose) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpos
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpose) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpose) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpose) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.6->cellpose) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.6
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.6->cellpose
100%|██████████| 2867/2867 [1:24:09<00:00,  1.76s/it]
Feature extraction complete. Results saved to /content/drive/My Drive/Senior Year/Applied Data Science/cellpose_features.csv
```

```python
import pandas as pd

cellpose_features_path = '/content/drive/My Drive/Senior Year/Applied Data Science/cellpose_features.csv'
metadata_path = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/metadata_BR00116991.csv'

cellpose_features = pd.read_csv(cellpose_features_path)
metadata_features = pd.read_csv(metadata_path)

cellpose_features['Trimmed_Image'] = cellpose_features['Image'].str[:9]

metadata_features['Trimmed_FileName'] = metadata_features['FileName_OrigRNA'].str[:9]

merged_features = pd.merge(
    cellpose_features,
    metadata_features[['Trimmed_FileName', 'Metadata_pert_iname']],
    left_on='Trimmed_Image',
    right_on='Trimmed_FileName',
    how='left'
).drop(columns=['Trimmed_FileName', 'Trimmed_Image'])  \

output_path = '/content/drive/My Drive/Senior Year/Applied Data Science/merged_features_metadata.csv'
merged_features.to_csv(output_path, index=False)

print(f"Merged dataset saved to {output_path}")
print(f"Final merged dataset has {len(merged_features)} rows.")
```

```
Merged dataset saved to /content/drive/My Drive/Senior Year/Applied Data Science/merged_features_metadata.csv
Final merged dataset has 396579 rows.
```

```python
import os
import pandas as pd
import tensorflow as tf

merged_features_path = '/content/drive/My Drive/Senior Year/Applied Data Science/merged_features_metadata.csv'
image_dir = '/content/drive/My Drive/Senior Year/Applied Data Science/Data/downsampled_data/'

merged_features = pd.read_csv(merged_features_path)

merged_features['Image'] = merged_features['Image'].apply(lambda x: x if x.endswith('.tiff') else x + '.tiff')

merged_features['Image'] = merged_features['Image'].apply(lambda x: os.path.join(image_dir, x))

merged_features = merged_features[merged_features['Image'].apply(os.path.exists)]

print(f"Number of images matched: {merged_features.shape[0]}")
```

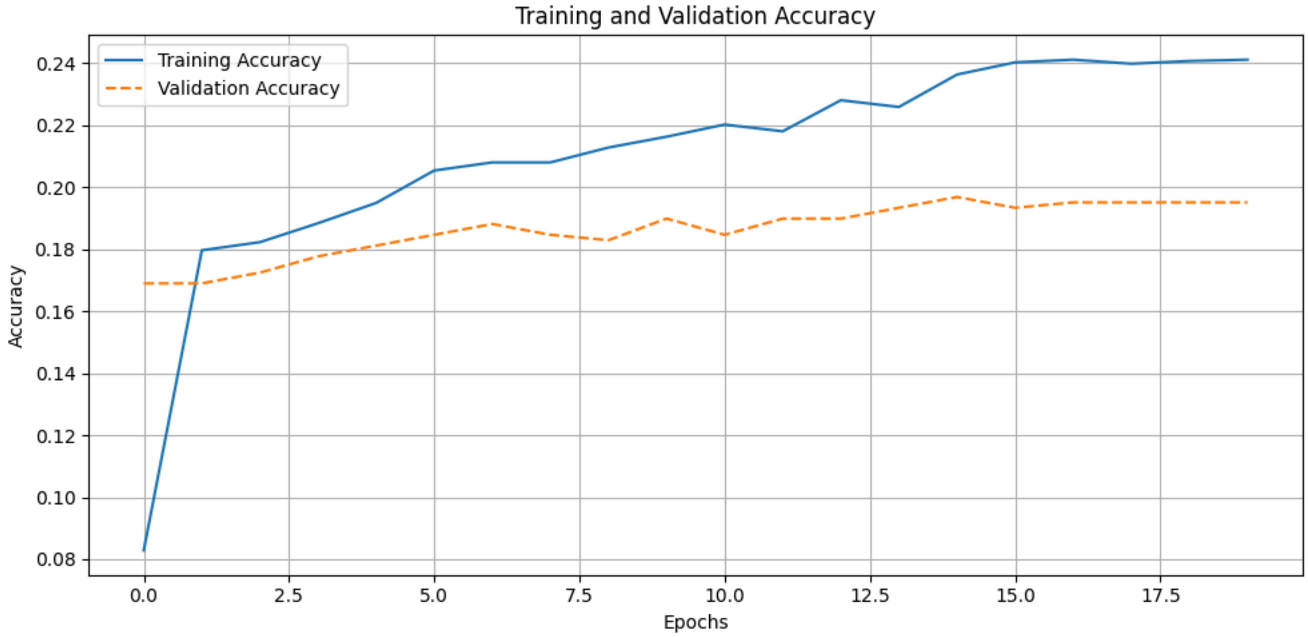
```
Number of images matched: 396579
```

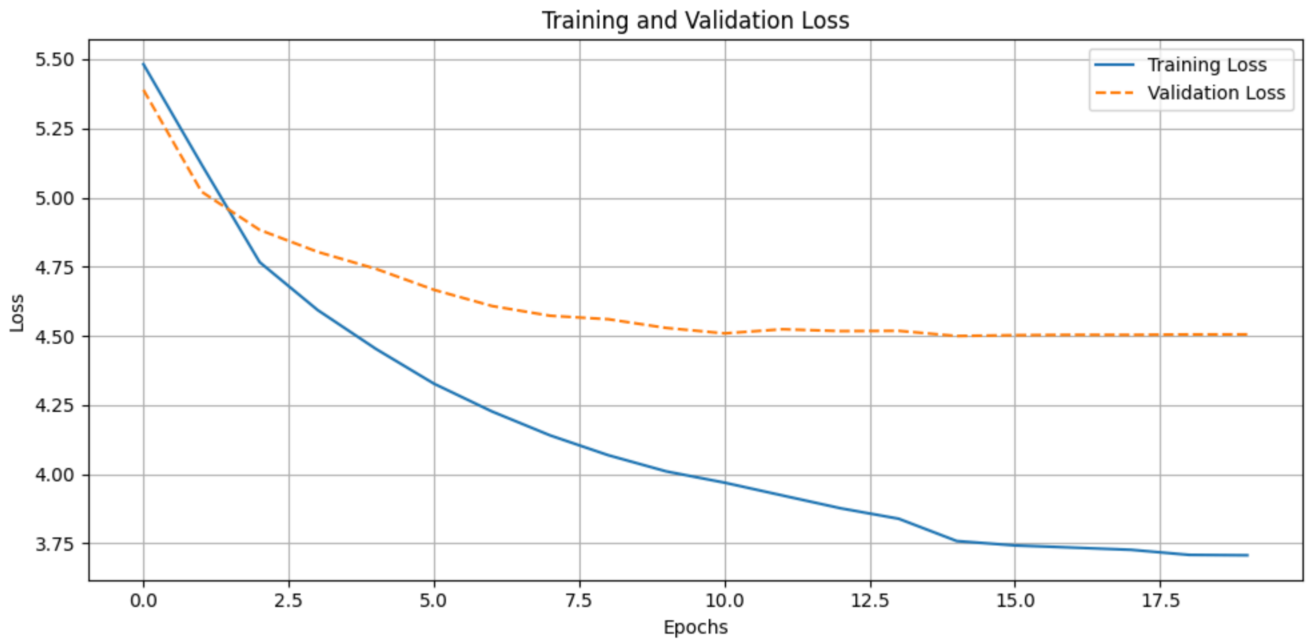
```
import os
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

merged_features_path = '/content/drive/My Drive/Senior Year/Applied Data Science/merged_features_metadata.csv'
merged_features = pd.read_csv(merged_features_path)

aggregated_features = merged_features.groupby('Image').agg({
    'Area': 'mean',
    'Mean Intensity': 'mean',
    'Bounding Box Sum': 'mean',
    'Eccentricity': 'mean',
    'Solidity': 'mean',
    'Metadata_pert_iname': 'first'
}).reset_index()

label_encoder = LabelEncoder()
aggregated_features['Encoded_Label'] = label_encoder.fit_transform(aggregated_features['Metadata_pert_iname'])

X = aggregated_features[['Area', 'Mean Intensity', 'Bounding Box Sum', 'Eccentricity', 'Solidity']].values
y = aggregated_features['Encoded_Label'].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(len(label_encoder.classes_), activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=30,
    batch_size=32,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=3, min_lr=1e-5)
    ],
    verbose=1
)

model.save('/content/drive/My Drive/Senior Year/Applied Data Science/aggregated_model.h5')

print("Training complete and model saved.")
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
72/72 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.0308 - loss: 5.5161 - val_accuracy: 0.1690 - val_loss: 5.3892 - learni
Epoch 2/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1802 - loss: 5.2335 - val_accuracy: 0.1690 - val_loss: 5.0211 - learnin
Epoch 3/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1710 - loss: 4.8335 - val_accuracy: 0.1725 - val_loss: 4.8830 - learnin
Epoch 4/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1773 - loss: 4.6448 - val_accuracy: 0.1777 - val_loss: 4.8035 - learnin
Epoch 5/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1920 - loss: 4.4562 - val_accuracy: 0.1812 - val_loss: 4.7424 - learnin
Epoch 6/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2061 - loss: 4.3107 - val_accuracy: 0.1847 - val_loss: 4.6666 - learnin
Epoch 7/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2088 - loss: 4.2251 - val_accuracy: 0.1882 - val_loss: 4.6080 - learnin
Epoch 8/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1926 - loss: 4.2008 - val_accuracy: 0.1847 - val_loss: 4.5728 - learnin
Epoch 9/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2144 - loss: 4.0175 - val_accuracy: 0.1829 - val_loss: 4.5603 - learnin
Epoch 10/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2281 - loss: 3.9656 - val_accuracy: 0.1899 - val_loss: 4.5288 - learnin
Epoch 11/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2243 - loss: 3.9069 - val_accuracy: 0.1847 - val_loss: 4.5091 - learnin
Epoch 12/30
```

```
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2155 - loss: 3.8868 - val_accuracy: 0.1899 - val_loss: 4.5241 - learnin
Epoch 13/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2197 - loss: 3.8816 - val_accuracy: 0.1899 - val_loss: 4.5174 - learnin
Epoch 14/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2266 - loss: 3.8524 - val_accuracy: 0.1934 - val_loss: 4.5183 - learnin
Epoch 15/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2346 - loss: 3.7359 - val_accuracy: 0.1969 - val_loss: 4.4996 - learnin
Epoch 16/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2412 - loss: 3.7286 - val_accuracy: 0.1934 - val_loss: 4.5026 - learnin
Epoch 17/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2374 - loss: 3.7498 - val_accuracy: 0.1951 - val_loss: 4.5040 - learnin
Epoch 18/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2372 - loss: 3.7501 - val_accuracy: 0.1951 - val_loss: 4.5038 - learnin
Epoch 19/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2458 - loss: 3.6578 - val_accuracy: 0.1951 - val_loss: 4.5054 - learnin
Epoch 20/30
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2293 - loss: 3.7287 - val_accuracy: 0.1951 - val_loss: 4.5055 - learnin
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
Training complete and model saved.
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linestyle='--')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('/content/drive/My Drive/Senior Year/Applied Data Science/training_validation_accuracy.png')
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss', linestyle='--')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('/content/drive/My Drive/Senior Year/Applied Data Science/training_validation_loss.png')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

```
import os
```