Butteryfly Species Identification Using CNN

In [1]:
```python
from data_loading import load_testing_data, get_key
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv


%load_ext autoreload
%autoreload 2
```

Loading and manipulating the data

The data for this project is from the "Butterfly Image Classification" dataset on Kaggle. [1]

In [2]:
```python
labels, images, images_seg, labels_dict = load_testing_data()
```

The images in this dataset were 224x224. We were unable to use a GPU, so training a model on images of that size would have taken a prohibitavily long time. We found that scaling the images down to 1/3 of their orignial resolution provided a good balance between speed and accurracy.

In [3]:
```python
scaled_images_seg = []
scaled_images = []
for img in images_seg:
    scaled_img_seg = cv.resize(img, None, fx= 0.3, fy= 0.3, interpolation= cv.INTER_LINEAR)
    scaled_images_seg.append(scaled_img_seg)

for img in images:
    scaled_img = cv.resize(img, None, fx= 0.3, fy= 0.3, interpolation= cv.INTER_LINEAR)
    scaled_images.append(scaled_img)
```

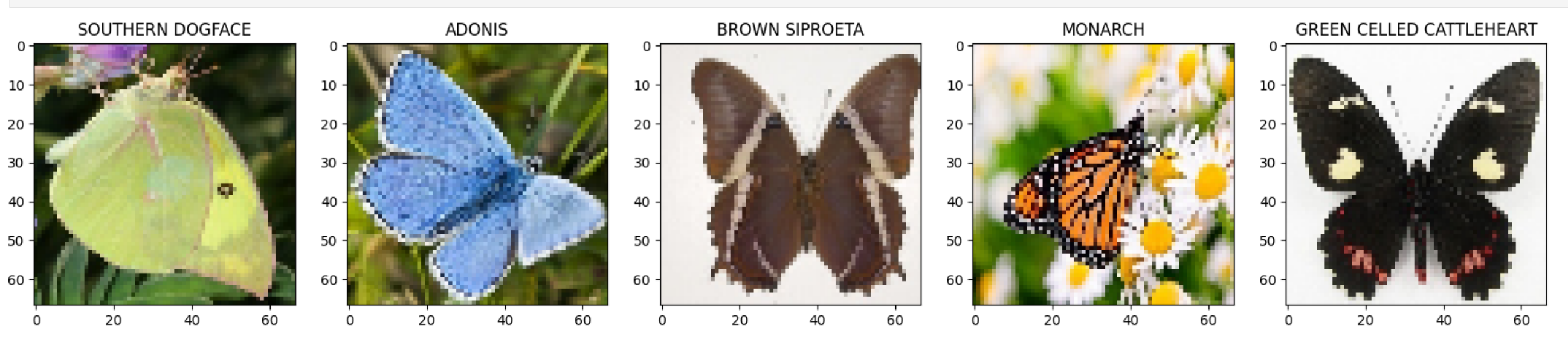
In [4]:
```python
print("Original Size: ", img.shape)
print("Scaled Size: ", scaled_img.shape)
```

```
Original Size:  (224, 224, 3)
Scaled Size:  (67, 67, 3)
```

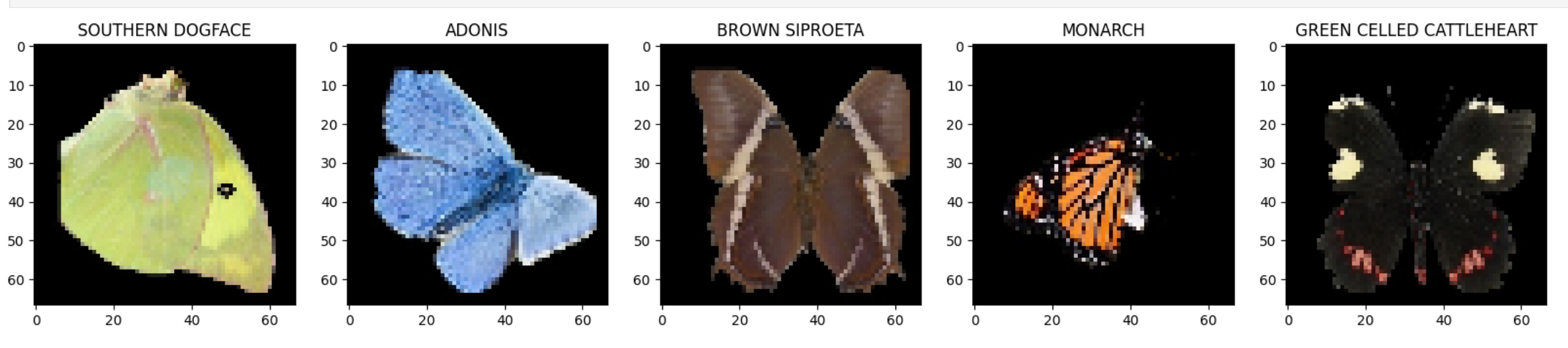
Displaying Sample Images

Since we want this CNN to be able to idenify butterfly species, we removed the background of the images. The code that accomplished this was sourced from an example notebook on Kaggle. [2] It uses OpenCV's grabCut method to separate the butterfly from the background. This allows the CNN to focus on the specifics of the butterfly and not any unintentional similarities found in background of the images. Samples of the original images and segmented images are found below.

In [5]:
```python
plt.figure(figsize=(20,5))                # Create a figure of size 20x5

# Create subplots of one row and three columns: subplot(nrows, ncols, index, **kwargs)
# For documentation, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
plt.subplot(1, 5, 1),plt.imshow(cv.cvtColor(scaled_images[0], cv.COLOR_BGR2RGB)),plt.title(get_key(0, labels_dict))
plt.subplot(1, 5, 2),plt.imshow(cv.cvtColor(scaled_images[1], cv.COLOR_BGR2RGB)),plt.title(get_key(1, labels_dict))
plt.subplot(1, 5, 3),plt.imshow(cv.cvtColor(scaled_images[2], cv.COLOR_BGR2RGB)),plt.title(get_key(2, labels_dict))
plt.subplot(1, 5, 4),plt.imshow(cv.cvtColor(scaled_images[3], cv.COLOR_BGR2RGB)),plt.title(get_key(3, labels_dict))
plt.subplot(1, 5, 5),plt.imshow(cv.cvtColor(scaled_images[4], cv.COLOR_BGR2RGB)),plt.title(get_key(4, labels_dict))
plt.show()
```



In [6]:
```python
plt.figure(figsize=(20,5))                # Create a figure of size 20x5

# Create subplots of one row and three columns: subplot(nrows, ncols, index, **kwargs)
# For documentation, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
plt.subplot(1, 5, 1),plt.imshow(cv.cvtColor(scaled_images_seg[0], cv.COLOR_BGR2RGB)),plt.title(get_key(0, labels_dict))
plt.subplot(1, 5, 2),plt.imshow(cv.cvtColor(scaled_images_seg[1], cv.COLOR_BGR2RGB)),plt.title(get_key(1, labels_dict))
plt.subplot(1, 5, 3),plt.imshow(cv.cvtColor(scaled_images_seg[2], cv.COLOR_BGR2RGB)),plt.title(get_key(2, labels_dict))
plt.subplot(1, 5, 4),plt.imshow(cv.cvtColor(scaled_images_seg[3], cv.COLOR_BGR2RGB)),plt.title(get_key(3, labels_dict))
plt.subplot(1, 5, 5),plt.imshow(cv.cvtColor(scaled_images_seg[4], cv.COLOR_BGR2RGB)),plt.title(get_key(4, labels_dict))
plt.show()
```

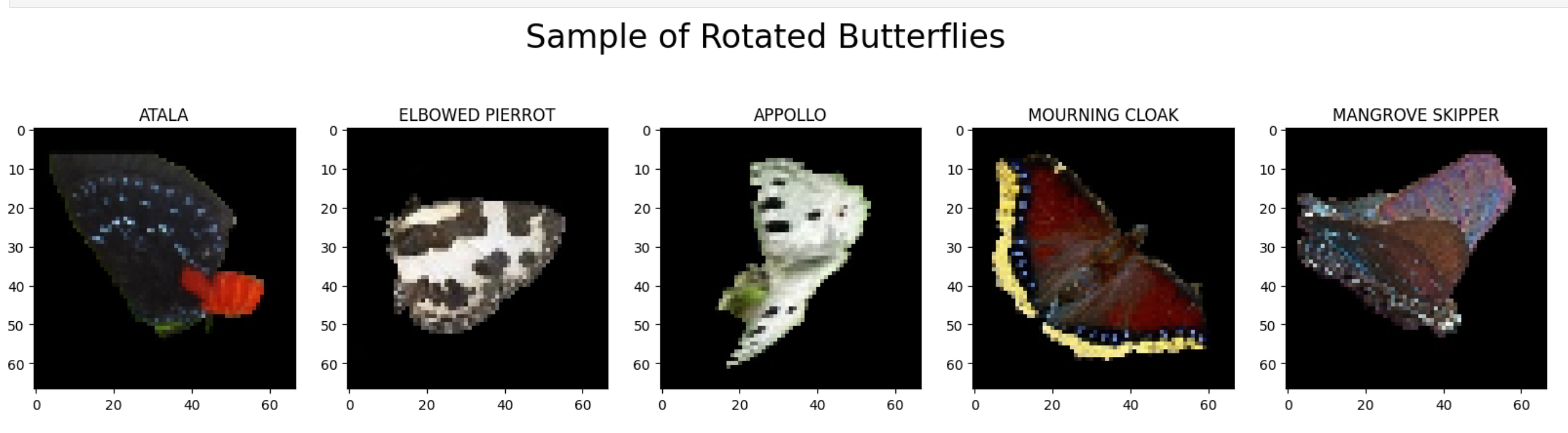


While training this model, we struggled to get a decent level of accuracy. There was only an average of 86 images per species in this dataset, which is limited. One trick we tried to improve accuracy was data augmentation. The application we tried was simple: rotating the butterflies 90 degrees clockwise and adding those images to the dataset. The hope was that the increased number of samples would increase accuracy. This effort was somewhat successful - the accuracy of the model increased from 56% without the augmented data to 62% with the augmented data. Interestingly, there was a repeatable decrease in accuracy when the images were flipped as well as flipped and rotated.

In [7]:
```python
index = 0
labels_seg = labels
while index < 6499:
    rotated_img = cv.rotate(scaled_images_seg[index], cv.ROTATE_90_CLOCKWISE)
    scaled_images_seg.append(rotated_img)
    label = labels[index]
    labels_seg.append(label)
    index += 1
```

In [8]:
```python
# index = 0
# labels_seg = labels
# while index < 6499:
#     # rotated_img = cv.rotate(scaled_images_seg[index], cv.ROTATE_90_CLOCKWISE)
#     rotated_img = cv.flip(rotated_img, -1)
#     scaled_images_seg.append(rotated_img)
#     label = labels[index]
#     labels_seg.append(label)
#     index += 1
```

In [9]:
```python
plt.figure(figsize=(20,5)), plt.suptitle("Sample of Rotated Butterflies", fontsize=24)

# Create subplots of one row and three columns: subplot(nrows, ncols, index, **kwargs)
# For documentation, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
plt.subplot(1, 5, 1),plt.imshow(cv.cvtColor(scaled_images_seg[-1], cv.COLOR_BGR2RGB)),plt.title(get_key(labels[-1][0], labels_dict))
plt.subplot(1, 5, 2),plt.imshow(cv.cvtColor(scaled_images_seg[-2], cv.COLOR_BGR2RGB)),plt.title(get_key(labels[-2][0], labels_dict))
plt.subplot(1, 5, 3),plt.imshow(cv.cvtColor(scaled_images_seg[-3], cv.COLOR_BGR2RGB)),plt.title(get_key(labels[-3][0], labels_dict))
plt.subplot(1, 5, 4),plt.imshow(cv.cvtColor(scaled_images_seg[-4], cv.COLOR_BGR2RGB)),plt.title(get_key(labels[-4][0], labels_dict))
plt.subplot(1, 5, 5),plt.imshow(cv.cvtColor(scaled_images_seg[-5], cv.COLOR_BGR2RGB)),plt.title(get_key(labels[-5][0], labels_dict))
plt.show()
```



Separating Training and Testing Data

The data is split into 70% training data and 30% testing data using sklearn's train_test_split fuction. It is then normalized and expanded to fit the dimensions expected by the model.

In [10]:
```python
images_train, images_test, labels_train, labels_test = train_test_split(scaled_images_seg, labels, test_size=0.3)
images_train = (np.expand_dims(images_train, axis=-1)/255.).astype(np.float32)
labels_train = np.array((labels_train)).astype(np.int64)
images_test = (np.expand_dims(images_test, axis=-1)/255.).astype(np.float32)
labels_test = np.array((labels_test)).astype(np.int64)
```

In [11]:
```python
plt.figure(figsize=(20,20)), plt.suptitle("Random Sampling of Segmented Butterfly Images from Augmented Dataset", fontsize=24)
random_inds = np.random.choice(1000,36)
for i in range(36):
    plt.subplot(6,6,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    image_ind = random_inds[i]
    plt.imshow(cv.cvtColor(np.squeeze(images_train[image_ind]),cv.COLOR_BGR2RGB), cmap=plt.cm.binary)
    plt.xlabel(get_key(labels_train[image_ind], labels_dict))
```

Building the Model

The network architecture of our model is as follows:

- INPUT: holds the raw image data of the detected size. In our case, [67x67x3].
- CONV 1: a convolutional layer of 50 filters with kernel size [3x3], resulting in an output volume of [65x65x50].
- RELU 1: applies the activation function. The output volume is unchanged.
- POOL 1: applies a pooling layer of size [2x2], resulting in an outpt volume of [32x32x50].
- CONV 2: a secon convolutional layer with 24 filters and kernel size [5x5]. The output volume is [28x28x24].
- RELU 2: applies the activation function. The output volume remains unchanged.
- POOL 2: applies another pooling layer of size [3x3], retulsing in an output volume of [9x9x24].
- FL: Flattens our [9x9x24] matrix into a vector of dimension 1944.
- FC 1: Reduces out 1944 units to 90.
- DROPOUT: Selects 30% of the neurons at random to drop out to prevent overfitting. The output volume is unchanged.
- FC 2: Computes the class scores, producing an output volume of [1x1x75], where the 75 numbers correspond to the class that image belongs to.
- The softmax activation function converts the resulting numbers into probabilitys between 0 and 1 that the image belongs to each class.

In [12]:
```python
def build_cnn_model():
    cnn_model = tf.keras.models.Sequential()

    # Define the first convolutional layer
    cnn_model.add(layers.Conv2D(filters=50, kernel_size=(3,3), activation='relu', input_shape=(images_train[0].shape[0:3])))


    # Define the first max pooling layer
    cnn_model.add(layers.MaxPool2D(pool_size=(2,2)))

    # Define the second convolutional layer
    cnn_model.add(layers.Conv2D(filters=24, kernel_size=(5,5), activation='relu'))

    # Define the second max pooling layer
    cnn_model.add(layers.MaxPool2D(pool_size=(3,3)))

    # Flatten the volume into a vector
    cnn_model.add(layers.Flatten())

    # A hidden fully connected layer
    cnn_model.add(layers.Dense(90, activation='relu'))

    # Add dropout layer
    cnn_model.add(layers.Dropout(0.3))

    # Define the last Dense layer to output the classification probabilities.
    # Pay attention to the activation needed a probability output
    cnn_model.add(layers.Dense(75, activation='softmax'))

    return cnn_model

cnn_model = build_cnn_model()

# Print the summary of the layers in the model.
print(cnn_model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 65, 65, 50)        1400

 max_pooling2d (MaxPooling2  (None, 32, 32, 50)        0
 D)

 conv2d_1 (Conv2D)           (None, 28, 28, 24)        30024

 max_pooling2d_1 (MaxPoolin  (None, 9, 9, 24)          0
 g2D)

 flatten (Flatten)           (None, 1944)              0

 dense (Dense)               (None, 90)                175050

 dropout (Dropout)           (None, 90)                0

 dense_1 (Dense)             (None, 75)                6825

=================================================================
Total params: 213299 (833.20 KB)
Trainable params: 213299 (833.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [13]:
```python
#Define the compile operation with your optimizer and learning rate of choice
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

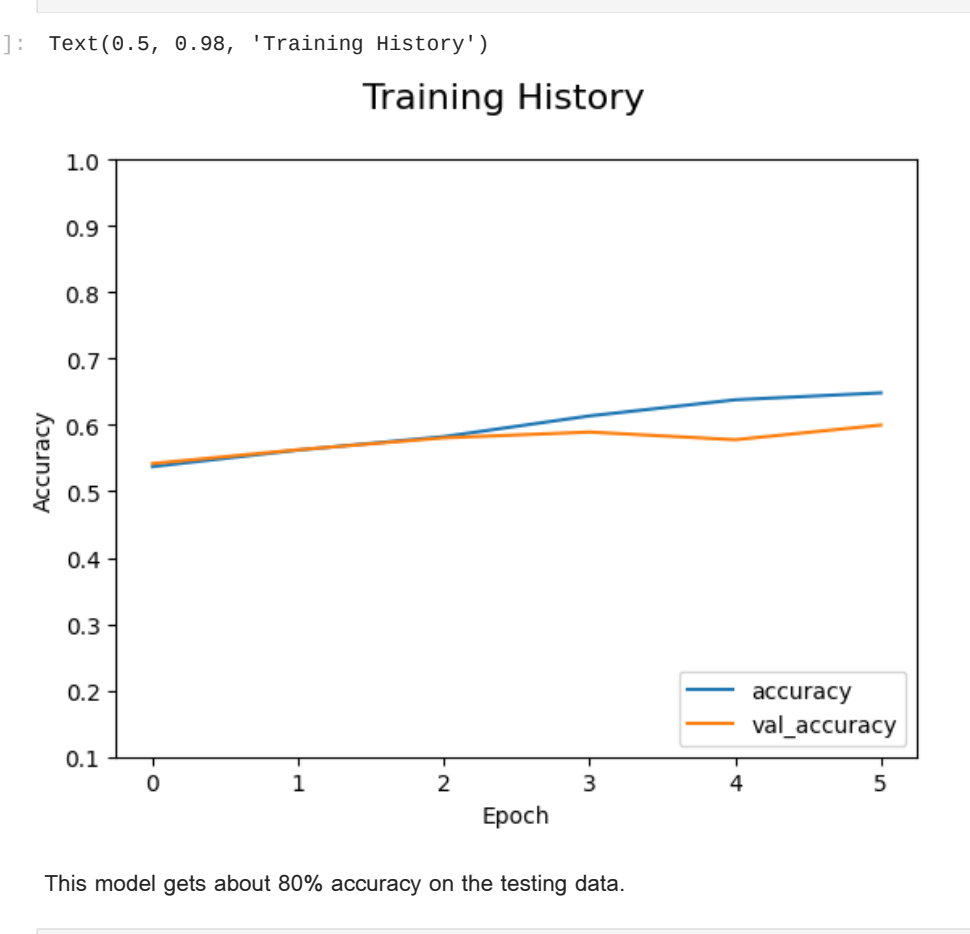
We did a lot of experimentation with batch size and number of epoch. Increasing the batch size and decreasing the number of epoch resulted in an increase in accuracy from 62% to 80% for a model. Although that model is saved, we were unable to replecate similar training results.

In [15]:
```python
# Define the batch size and the number of epochs to use during training
BATCH_SIZE = 75
EPOCHS = 6

# Use model.fit to train the CNN model, with the same batch_size and number of epochs previously used.
history = cnn_model.fit(images_train, labels_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
                        validation_data=(images_test, labels_test))
```

```
Epoch 1/6
122/122 [==============================] - 30s 249ms/step - loss: 1.6131 - accuracy: 0.5372 - val_loss: 1.6606 - val_accuracy: 0.5418
Epoch 2/6
122/122 [==============================] - 29s 234ms/step - loss: 1.5027 - accuracy: 0.5621 - val_loss: 1.5849 - val_accuracy: 0.5626
Epoch 3/6
122/122 [==============================] - 29s 235ms/step - loss: 1.4081 - accuracy: 0.5823 - val_loss: 1.5267 - val_accuracy: 0.5805
Epoch 4/6
122/122 [==============================] - 30s 243ms/step - loss: 1.3004 - accuracy: 0.6135 - val_loss: 1.4951 - val_accuracy: 0.5892
Epoch 5/6
122/122 [==============================] - 29s 242ms/step - loss: 1.2215 - accuracy: 0.6378 - val_loss: 1.5336 - val_accuracy: 0.5777
Epoch 6/6
122/122 [==============================] - 28s 229ms/step - loss: 1.1496 - accuracy: 0.6483 - val_loss: 1.4654 - val_accuracy: 0.5997
```

In [16]:
```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')
plt.suptitle("Training History", fontsize=16)
```

Out[16]:
```
Text(0.5, 0.98, 'Training History')
```



This model gets about 80% accuracy on the testing data.

In [17]:
```python
# Evaluating the model on the testing data
test_loss, test_acc = cnn_model.evaluate(images_test, labels_test)

print('Test accuracy:', test_acc)
```

```
122/122 [==============================] - 3s 25ms/step - loss: 1.4654 - accuracy: 0.5997
Test accuracy: 0.5997436046600342
```

Other methods for evaluating the model

The model can then be saved after training.

In [18]:
```python
# model_name = "finalModel"
# cnn_model.save("./cnn_models/" + model_name)
```

Specify which model to load, then load the model.

In [19]:
```python
model_name = "finalModel"
model = keras.models.load_model("./cnn_models/" + model_name)
```

Use the model to predict the species of the butterflies in the test images.

In [20]:
```python
predictions = model.predict(images_test)
predicted_labels = []

for prediction in predictions:
    index = np.argmax(prediction)
    label = get_key(index, labels_dict)
    predicted_labels.append(label)
```

```
122/122 [==============================] - 3s 25ms/step
```

The following shows the classification report for the predictions made by this model.

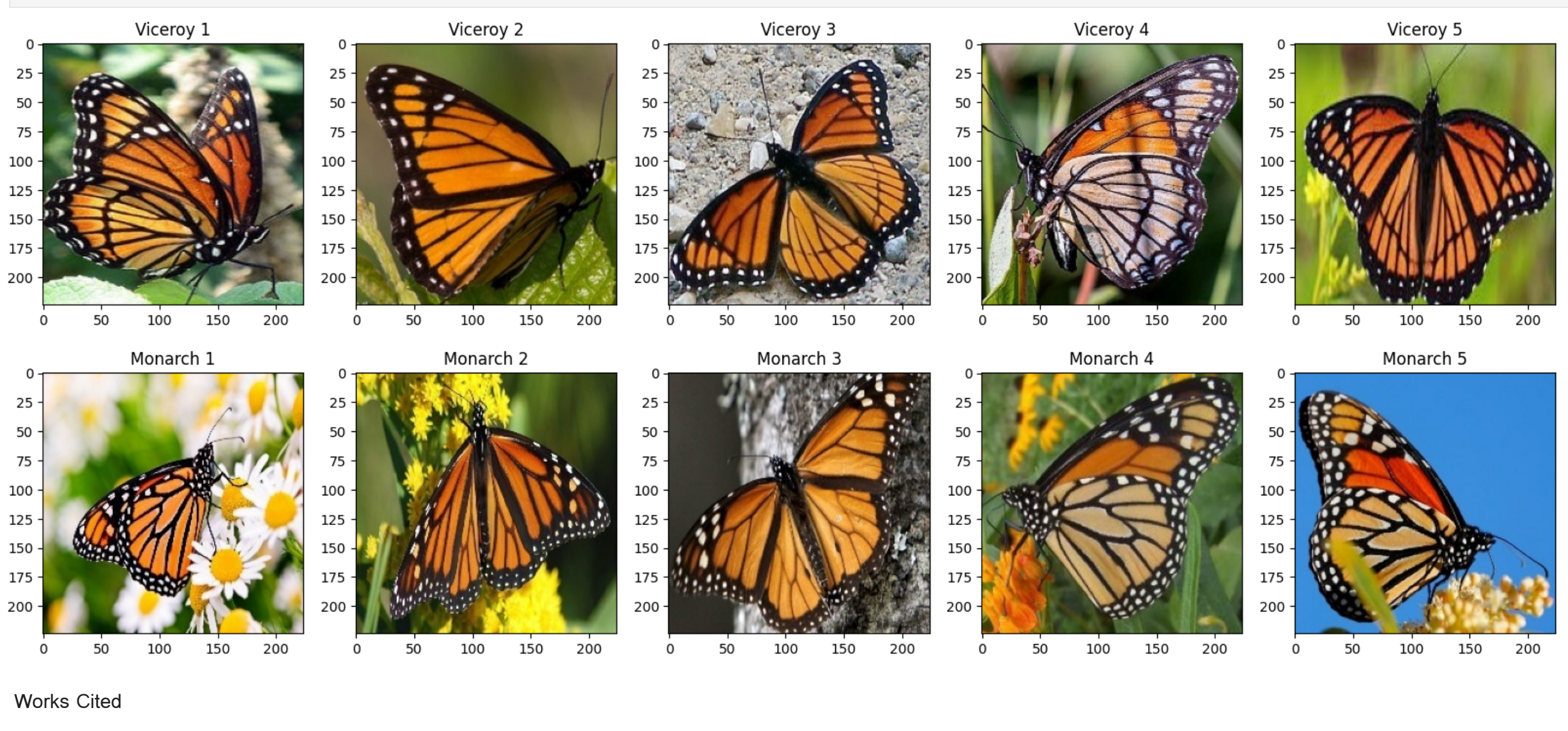
In [21]:
```python
from sklearn.metrics import classification_report
labels_test_string = []
for label in labels_test:
    string_label = get_key(label, labels_dict)
    labels_test_string.append(string_label)

species = np.unique(predicted_labels)
print(classification_report(labels_test_string, predicted_labels))
```

```
                           precision    recall  f1-score   support

                   ADONIS       1.00      0.93      0.96        55
AFRICAN GIANT SWALLOWTAIL       0.87      0.89      0.88        45
           AMERICAN SNOOT       0.95      0.86      0.90        43
                    AN 88       1.00      0.98      0.99        55
                  APPOLLO       0.91      0.82      0.86        61
                    ATALA       0.98      1.00      0.99        60
 BANDED ORANGE HELICONIAN       0.88      0.93      0.90        54
           BANDED PEACOCK       0.89      0.85      0.87        48
            BECKERS WHITE       0.92      0.69      0.79        52
         BLACK HAIRSTREAK       0.88      0.96      0.91        45
              BLUE MORPHO       0.77      0.97      0.86        34
        BLUE SPOTTED CROW       0.88      0.92      0.90        50
           BROWN SIPROETA       0.90      0.97      0.93        63
            CABBAGE WHITE       0.80      0.95      0.87        62
          CAIRNS BIRDWING       0.95      0.96      0.95        54
      CHECQUERED SKIPPER       0.93      0.90      0.91        41
                 CHESTNUT       0.93      0.90      0.91        41
                CLEOPATRA       0.86      0.88      0.87        65
       CLODIUS PARNASSIAN       0.76      0.77      0.77        53
          CLOUDED SULPHUR       0.63      0.57      0.60        51
        COMMON BANDED AWL       0.84      0.84      0.84        50
        COMMON WOOD-NYMPH       0.61      0.82      0.70        34
              COPPER TAIL       0.78      0.83      0.80        54
                  CRECENT       0.85      0.94      0.89        65
            CRIMSON PATCH       1.00      0.97      0.98        31
           DANAID EGGFLY       0.85      0.92      0.88        48
             EASTERN COMA       0.88      0.78      0.83        65
     EASTERN DAPPLE WHITE       0.88      0.89      0.88        55
       EASTERN PINE ELFIN       0.96      0.83      0.89        66
          ELBOWED PIERROT       0.93      1.00      0.96        52
              GOLD BANDED       0.97      0.80      0.88        44
             GREAT EGGFLY       0.93      0.89      0.91        45
                GREAT JAY       0.91      0.95      0.93        61
 GREEN CELLED CATTLEHEART       0.94      0.91      0.92        54
          GREY HAIRSTREAK       0.84      0.88      0.86        49
            INDRA SWALLOW       0.92      0.85      0.89        41
          IPHICLUS SISTER       0.97      0.92      0.94        64
                    JULIA       0.72      1.00      0.84        44
             LARGE MARBLE       0.89      0.75      0.82        53
                MALACHITE       0.90      0.95      0.92        37
         MANGROVE SKIPPER       0.92      0.84      0.88        57
                   MESTRA       0.87      0.94      0.90        48
                METALMARK       0.98      0.88      0.92        48
   MILBERTS TORTOISESHELL       0.93      0.94      0.93        68
                  MONARCH       0.92      0.94      0.93        48
           MOURNING CLOAK       0.96      0.97      0.97        78
           ORANGE OAKLEAF       0.85      0.88      0.87        52
               ORANGE TIP       1.00      0.96      0.98        52
           ORCHARD SWALLOW       0.84      0.86      0.85        44
             PAINTED LADY       0.96      0.82      0.88        56
               PAPER KITE       0.94      1.00      0.97        48
                  PEACOCK       0.98      0.98      0.98        53
               PINE WHITE       0.76      0.89      0.82        46
         PIPEVINE SWALLOW       0.89      0.89      0.89        53
                 POPINJAY       0.90      0.98      0.94        48
        PURPLE HAIRSTREAK       0.83      0.81      0.82        47
          PURPLISH COPPER       0.75      0.79      0.77        53
            QUESTION MARK       0.85      0.70      0.77        47
              RED ADMIRAL       0.92      0.98      0.95        49
              RED CRACKER       0.91      0.94      0.92        63
              RED POSTMAN       0.96      0.94      0.95        50
       RED SPOTTED PURPLE       0.84      0.88      0.86        60
           SCARCE SWALLOW       0.99      0.96      0.97        69
      SILVER SPOT SKIPPER       0.95      0.86      0.90        49
            SLEEPY ORANGE       0.73      0.84      0.78        70
                SOOTYWING       0.96      0.86      0.91        51
         SOUTHERN DOGFACE       0.82      0.71      0.76        58
           STRAITED QUEEN       0.91      0.89      0.90        47
        TROPICAL LEAFWING       0.88      0.75      0.81        48
       TWO BARRED FLASHER       0.90      0.96      0.93        49
                   ULYSES       0.97      0.97      0.97        59
                  VICEROY       0.87      0.89      0.88        44
               WOOD SATYR       0.91      0.82      0.86        50
       YELLOW SWALLOW TAIL       0.95      0.93      0.94        41
          ZEBRA LONG WING       1.00      0.94      0.97        53

                 accuracy                           0.89      3900
                macro avg       0.89      0.89      0.89      3900
             weighted avg       0.89      0.89      0.89      3900
```

Conclusion

We were able to achieve a 89% accuracy rate with our final model, which we were very happy with. However, there is room for improvement. Several of the species appear very similar, such as the Monarch and Viceroy pictured below. It may be possible to improve accuracy by grouping similar butterfly species in a way that their class numbers are adjacent. The current set up assigns class number based on the order in which the files are read.

In [22]:
```python
from data_loading import get_val

vals, count = np.unique(labels, return_inverse=False,
                              return_counts=True)

indices = []
for val in vals:
    index = np.where(labels == val)
    indices.append(index)

v_val = get_val("VICEROY", labels_dict)
m_val = get_val("MONARCH", labels_dict)

viceroys = []
i=0
for i in range(5):
    index = indices[v_val][0][i]
    image = images[index]
    viceroys.append(image)

plt.figure(figsize=(20,5))                # Create a figure of size 20x5

# Create subplots of one row and three columns: subplot(nrows, ncols, index, **kwargs)
# For documentation, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
plt.subplot(1, 5, 1),plt.imshow(cv.cvtColor(viceroys[0], cv.COLOR_BGR2RGB)),plt.title("Viceroy 1")
plt.subplot(1, 5, 2),plt.imshow(cv.cvtColor(viceroys[1], cv.COLOR_BGR2RGB)),plt.title("Viceroy 2")
plt.subplot(1, 5, 3),plt.imshow(cv.cvtColor(viceroys[2], cv.COLOR_BGR2RGB)),plt.title("Viceroy 3")
plt.subplot(1, 5, 4),plt.imshow(cv.cvtColor(viceroys[3], cv.COLOR_BGR2RGB)),plt.title("Viceroy 4")
plt.subplot(1, 5, 5),plt.imshow(cv.cvtColor(viceroys[4], cv.COLOR_BGR2RGB)),plt.title("Viceroy 5")
plt.show()

monarchs = []
i=0
for i in range(5):
    index = indices[m_val][0][i]
    image = images[index]
    monarchs.append(image)

plt.figure(figsize=(20,5))                # Create a figure of size 20x5

# Create subplots of one row and three columns: subplot(nrows, ncols, index, **kwargs)
# For documentation, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
plt.subplot(1, 5, 1),plt.imshow(cv.cvtColor(monarchs[0], cv.COLOR_BGR2RGB)),plt.title("Monarch 1")
plt.subplot(1, 5, 2),plt.imshow(cv.cvtColor(monarchs[1], cv.COLOR_BGR2RGB)),plt.title("Monarch 2")
plt.subplot(1, 5, 3),plt.imshow(cv.cvtColor(monarchs[2], cv.COLOR_BGR2RGB)),plt.title("Monarch 3")
plt.subplot(1, 5, 4),plt.imshow(cv.cvtColor(monarchs[3], cv.COLOR_BGR2RGB)),plt.title("Monarch 4")
plt.subplot(1, 5, 5),plt.imshow(cv.cvtColor(monarchs[4], cv.COLOR_BGR2RGB)),plt.title("Monarch 5")
plt.show()
```



Works Cited

[1] Depie. (September 2023). Butterfly Image Classification. (Version 1) [Dataset]. Available: https://www.kaggle.com/datasets/phucthaiv02/butterfly-image-classification.

[2] P. Panjabud. (October 2023). Butterfly - GrabCut + DenseNet(CNN). (Version 5.0) [Source Code]. Available: https://www.kaggle.com/code/pacawat/butterfly-grabcut-densenet-cnn