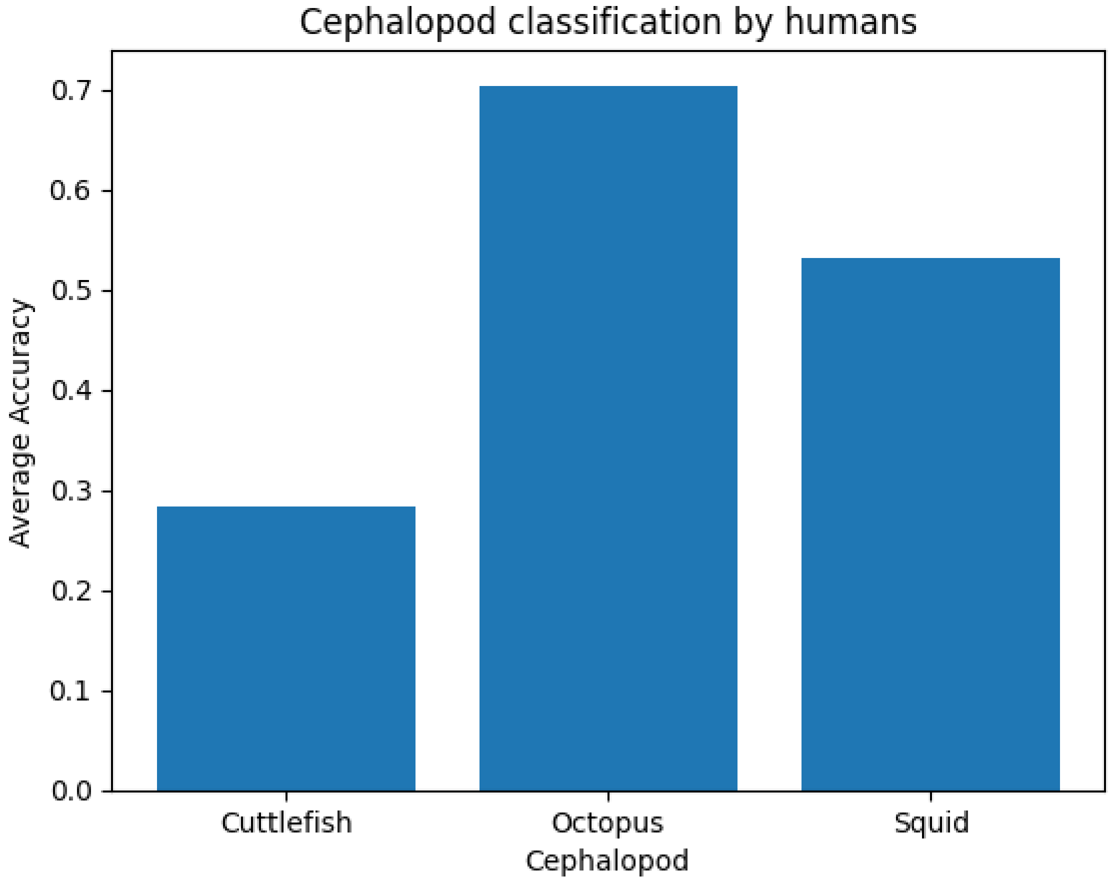
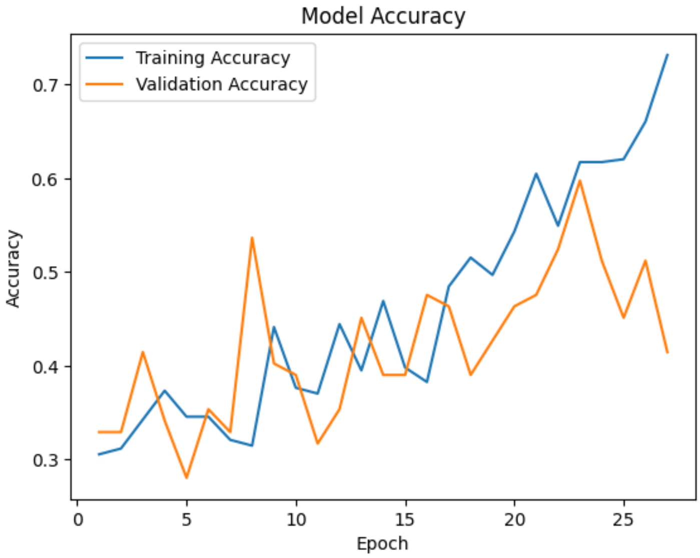
Jason, 1C, Cephalopod Image Classification Model

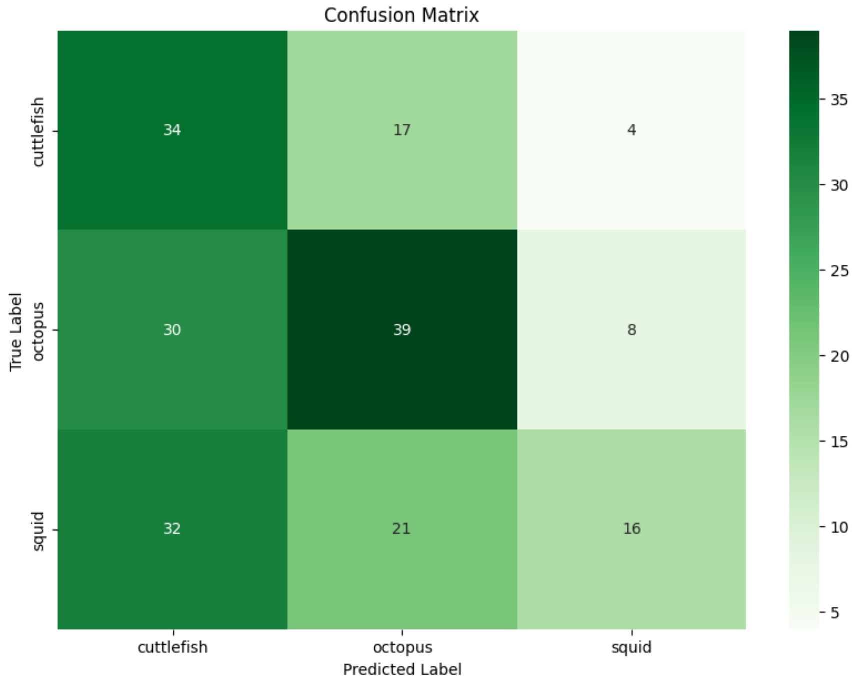
First of all, I created a form to calculate the performances for humans when they classify cephalopods. The average accuracies per cephalopod can be found below. This performance will be used as a basis for what our model should improve upon.



After this, I created a model with not much depth, to also use it as a baseline for what a model with more depth should improve upon too. In the scores below you can see that the performances weren’t amazing and had a lot of overfitting.


Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.3
0.4
0.5
0.6
0.7
0
5
10
15
20
25
Epoch


Model Loss
Training Loss
Validation Loss
Loss
1
2
3
4
5
6
0
5
10
15
20
25
Epoch



After this, I created multiple versions of the model: the first one didn’t have much preprocessing and also didn’t have too much depth. This model improved just slightly better than the first model with barely any depth. After this I created a model, which got generated images for training data. This data would be generated to be flipped, rotated, have a different brightness, etc. This model also improved slightly better than the past model, so I created a new model. The third model trains on the normal dataset while also using a pre-trained model. The performances for this went up towards 85% without much overfitting. I eventually came to the final model, which had the second and third model’s techniques combined, which I will show below.

First I apply data augmentation:

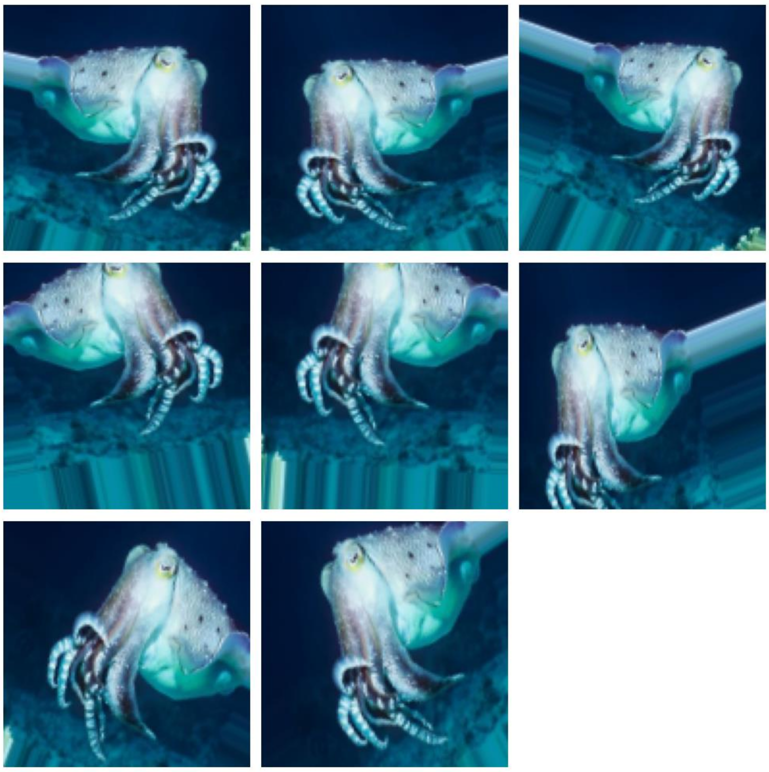
Afterwards I create the image generator so we can easily apply image augmentation. I use the same parameters as b
desired amount of images.

```
# Create image generator.
V4_data_gen = ImageDataGenerator(rotation_range=30,
                                 width_shift_range=0.2,
                                 height_shift_range=0.2,
                                 horizontal_flip=True,
                                 zoom_range=0.2)

# Function for image generator.
def GenerateImages(generator, image, num_images):
    new_images = []
    # Reshape images
    image = np.reshape(image, (1,) + image.shape)
    # Loop through images
    for _ in range(num_images):
        batch = next(generator.flow(image, batch_size=1))
        new_images.append(batch[0])
    # Return new images.
    return np.array(new_images)
```

Next I will apply augmentation on all the images. I chose for 8 variants again to make sure I have a varied dataset.

```
# Create a list
V4_X_aug = []
for i in range(len(V4_X_resize)):
    aug = GenerateImages(data_gen, V4_X_resize[i], 8)
    V4_X_aug.append(aug)
```

This will result in below images being generated:



After this, I split the dataset into training and testing data:

```python
# Create LabelEncoder
label_encoder = LabelEncoder()

y_enc = label_encoder.fit_transform(y)

# Reshape all the data
V4_y_cat = to_categorical(y_enc, num_classes=3)

# Apply flattening on the augmented images.
V4_X_aug_2 = list(itertools.chain(*V4_X_aug))
V4_np_X = np.array(V4_X_aug_2)
# Repeat the labels to match augmentation
V4_y_cat_aug = np.repeat(np.array(V4_y_cat), repeats=8, axis=0)

# Create testing and training sets
V4_X_train, V4_X_test, V4_y_train, V4_y_test = train_test_split(V4_np_X, V4_y_cat_aug, test_size=0.3, random_state=0)

V4_X_train, V4_X_val, V4_y_train, V4_y_val = train_test_split(V4_X_train, V4_y_train, test_size=0.3, random_state=0)
```

Now to finally train the model, I would first have to get the pre-trained model:

```python
# Set the base model for transfer learning.
V4_base_model = keras.applications.MobileNet(
                include_top=False,
                weights='imagenet',
                input_shape=V4_X_train.shape[1:])
# Print out the summary of the base model.
V4_base_model.summary()
```

After this I create the model's structure:

```python
# Initialize model
V4_model = Sequential([
    Flatten(input_shape=(4, 4, 1024)),
    Dense(10, activation='relu'),
    Dropout(rate=0.2),
    Dense(3, activation="softmax")
])
V4_model.summary()
```

And eventually, I will use the pre-trained model to look for certain important features in the images and then compile the model:

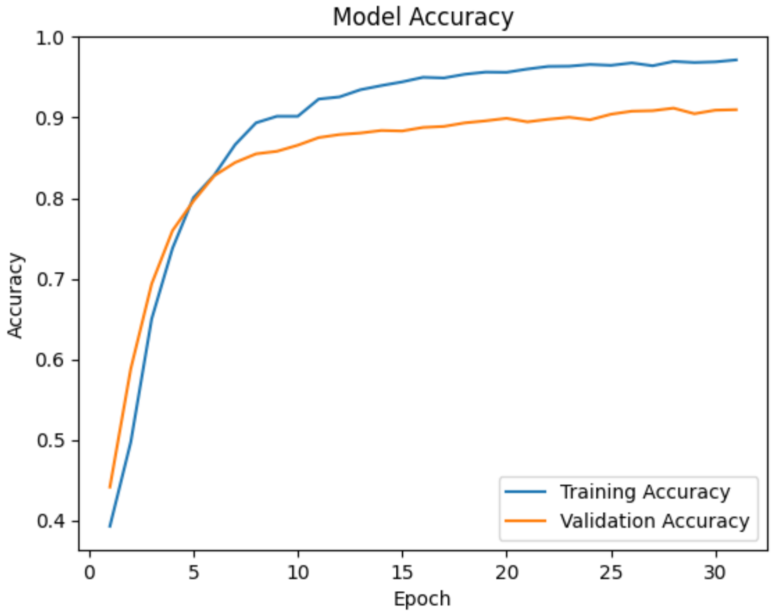
```python
# Preprocess the images.
V4_preproces_X_train = keras.applications.mobilenet.preprocess_input(V4_X_train * 255)
# Predict the features.
V4_output_features_train = V4_base_model.predict(V4_preproces_X_train)
# Now do the same for the validation set.
V4_preproces_X_val = keras.applications.mobilenet.preprocess_input(V4_X_val * 255)
V4_output_features_val = V4_base_model.predict(V4_preproces_X_val)
# And finally the same for testing sets.
V4_preproces_X_test = keras.applications.mobilenet.preprocess_input(V4_X_test * 255)
V4_output_features_test = V4_base_model.predict(V4_preproces_X_test)

# Set optimizer
V4_opt = Adam(learning_rate=0.0001)
# Finally, compile the model.
V4_model.compile(optimizer=V4_opt, loss="categorical_crossentropy", metrics=["accuracy"])
```

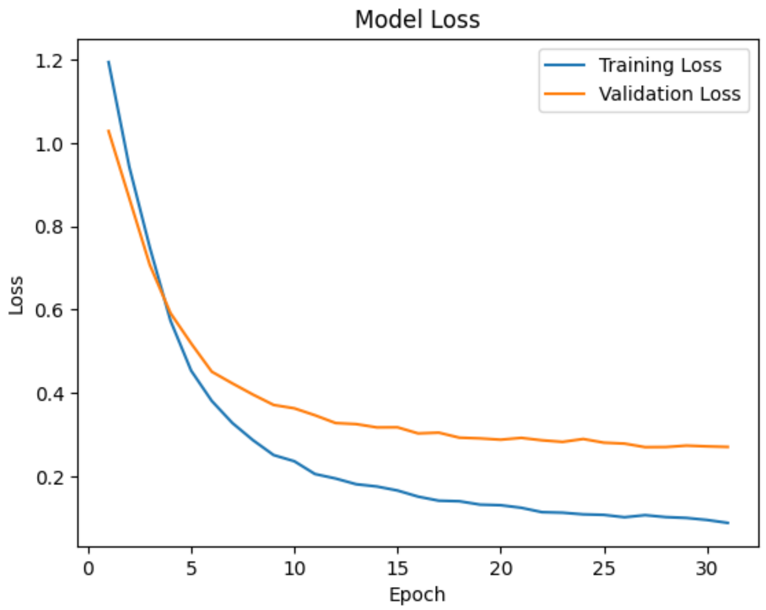
```
116/116 [==============================] - 24s 194ms/step
50/50 [==============================] - 12s 230ms/step
71/71 [==============================] - 15s 209ms/step
```

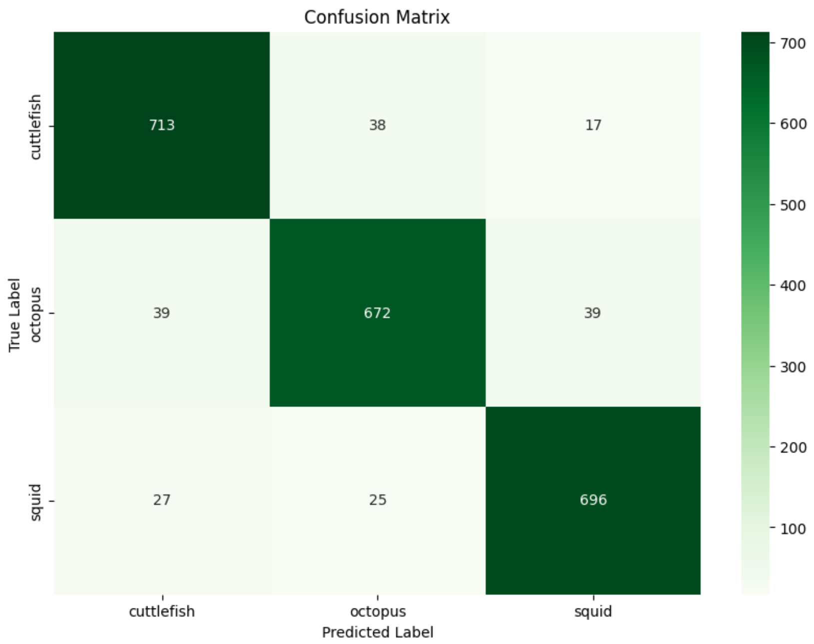
After this, I train the model using EarlyStopping, which will stop the model from training when no more improvements are made in the performances

```python
# Use early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True)
# Train the model.
V4_history = V4_model.fit(V4_output_features_train, V4_y_train, epochs=100, batch_size=256, validation_data=(V4_output_features_val, V4_y_val), callbacks=[early_stopping])
```

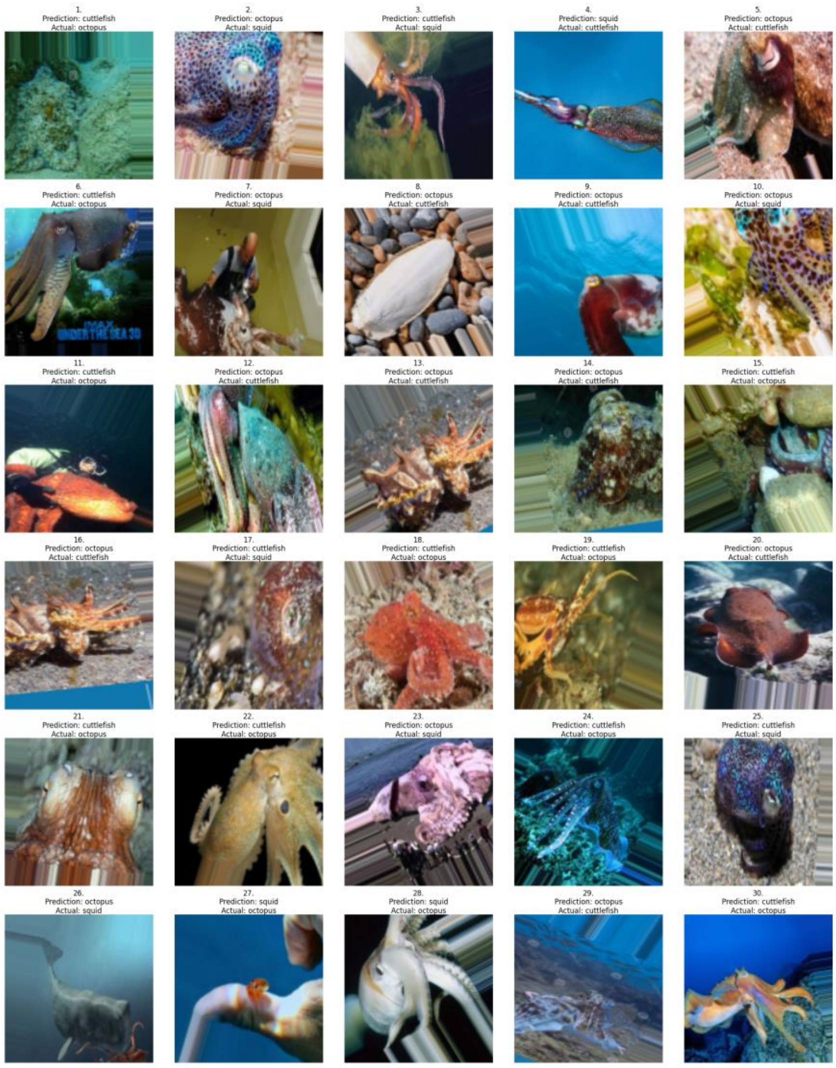
And eventually, we can see the performances for this model, which now has less overfitting and a much better accuracy:







Finally, we evaluate why the model made certain errors, in the error analysis phase:



In this phase, I evaluate why the model made the wrong predictions, which could be due to mislabeling, similarities between species, or just images on which the subject is hard to find.