CSE 428 Milestone-2

Aabrar Islam - 20101361

Ayen Aziza Haque - 20301487

Simin Waliza - 20101401

Zahin Shabab - 20101165

Shahriar Azad Frahim- 20101223

Augmentation

Types of Augmentations used:

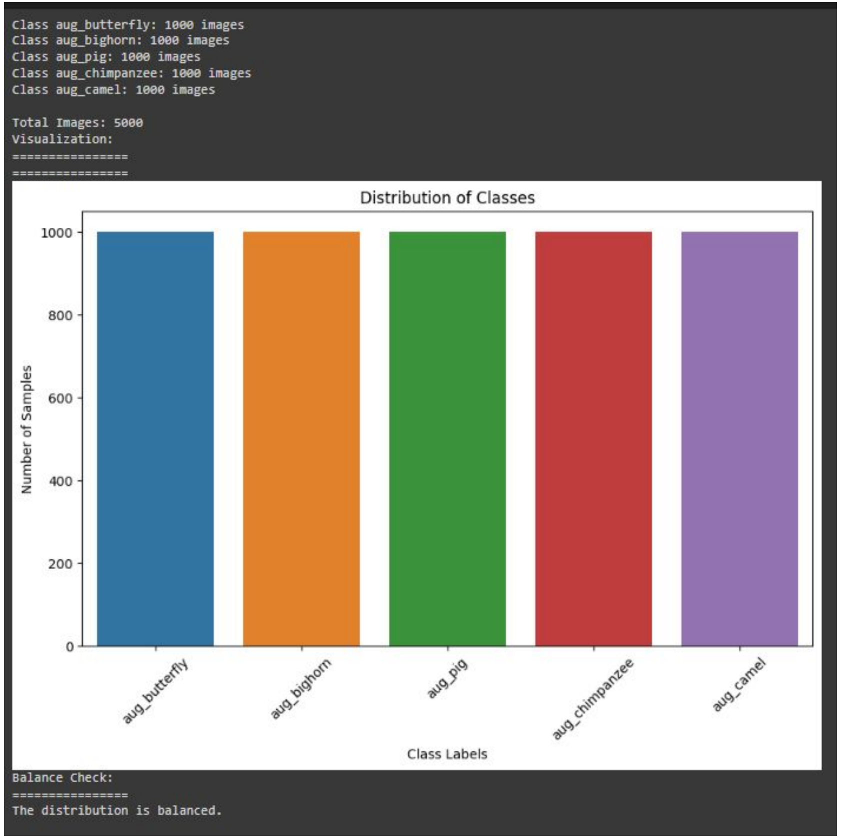
1. Rotation(left and right)
2. Flipping(left and right)
3. Zooming(in and out)
4. Changing contrast
5. Skewing

Augmentor

```
def augmentor(input_dir, output_dir, desired_count):
    p = Augmentor.Pipeline(input_dir)
    p.rotate(probability=0.3, max_left_rotation=10, max_right_rotation=10)
    p.flip_left_right(probability=0.38)
    p.zoom(probability=0.2, min_factor=1.1, max_factor=1.5)
    p.skew(probability=0.3, magnitude=0.3)
    p.random_contrast(probability=0.3, min_factor=0.8, max_factor=1.2)
    original_count = len(p.augmentor_images)
    images_needed = desired_count - original_count
    while images_needed > 0:
        p.sample(1)
        images_needed -= 1
    for root, _, files in os.walk(input_dir):
        for filename in files:
            source_path = os.path.join(root, filename)
            if filename.startswith("output_"):
                destination_path = os.path.join(output_dir, filename)
                if not os.path.exists(destination_path):
                    shutil.move(source_path, destination_path)
            else:
                destination_path = os.path.join(output_dir, filename)
                if not os.path.exists(destination_path):
                    shutil.copy(source_path, destination_path)
parent_folder = '/content/drive/MyDrive/428/Group_2/test'
output_parent_folder = '/content/drive/MyDrive/428/Group_2/aug_test'
```

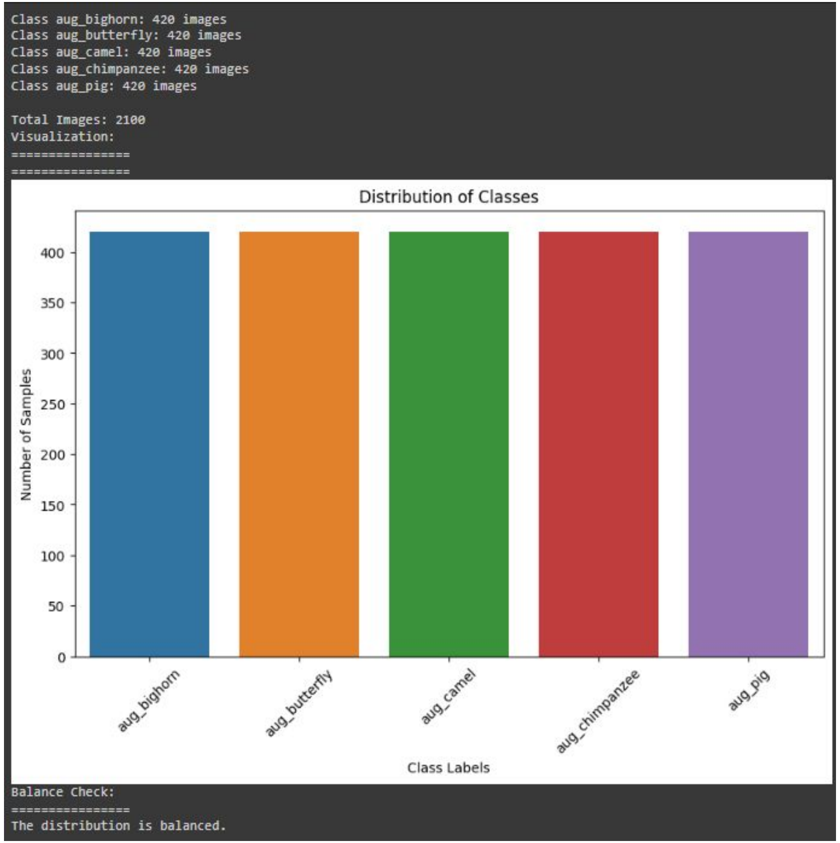
Augmented Train Dataset

- 5 classes
- Train(5000)
  - Bighorn=1000
  - Butterfly=1000
  - Camel=1000
  - Chimpanzee=1000
  - Pig=1000



Augmented Test Dataset

- 5 classes
- Test(2100)
  - Bighorn=420
  - Butterfly=420
  - Camel=420
  - Chimpanzee=420
  - Pig=420



Performance Metric Formulae

1. **<u>Sensitivity:</u>** True Positive / (True Positive + False Negative)
2. **<u>Specificity :</u>** True Negative / (True Negative + False Positive)
3. **<u>Positive Predictive Value(PPV):</u>** True Positive / (True Positive + False Positive)
4. **<u>Negative Predictive Value(NPV):</u>** True Negative / (True Negative + False Negative)
5. **<u>F Score:</u>**(2*Precision*Recall) / (Precision + Recall)

```python
def evaluate(model, X, y):
    y_pred_probs = model.predict(X)
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = np.argmax(y, axis=1)
    accuracy = accuracy_score(y_true, y_pred)
    conf_matrix = confusion_matrix(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average='weighted')
    sensitivity, specificity, PPV, NPV = [], [], [], []
    for i in range(len(conf_matrix)):
        TP = conf_matrix[i, i]
        FN = conf_matrix[i, :].sum() - TP
        FP = conf_matrix[:, i].sum() - TP
        TN = conf_matrix.sum() - (TP + FN + FP)

        sensitivity.append(TP / (TP + FN) if (TP + FN) != 0 else 0)
        specificity.append(TN / (TN + FP) if (TN + FP) != 0 else 0)
        PPV.append(TP / (TP + FP) if (TP + FP) != 0 else 0)
        NPV.append(TN / (TN + FN) if (TN + FN) != 0 else 0)

    print("Accuracy:", accuracy)
    print("Sensitivity (per class):", sensitivity)
    print("Specificity (per class):", specificity)
    print("Positive Predictive Value (PPV) (per class):", PPV)
    print("Negative Predictive Value (NPV) (per class):", NPV)
    print("F1 Score:", f1)
    plot_confusion_matrix(conf_matrix)
```

Resnet50

```python
def identity_block(X, f, filters):
    F1, F2, F3 = filters

    X_shortcut = X

    X = layers.Conv2D(F1, (1, 1), strides=(1, 1), padding='valid')(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(F2, (f, f), strides=(1, 1), padding='same')(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(F3, (1, 1), strides=(1, 1), padding='valid')(X)
    X = layers.BatchNormalization(axis=3)(X)

    X = layers.add([X, X_shortcut])
    X = layers.Activation('relu')(X)

    return X
```

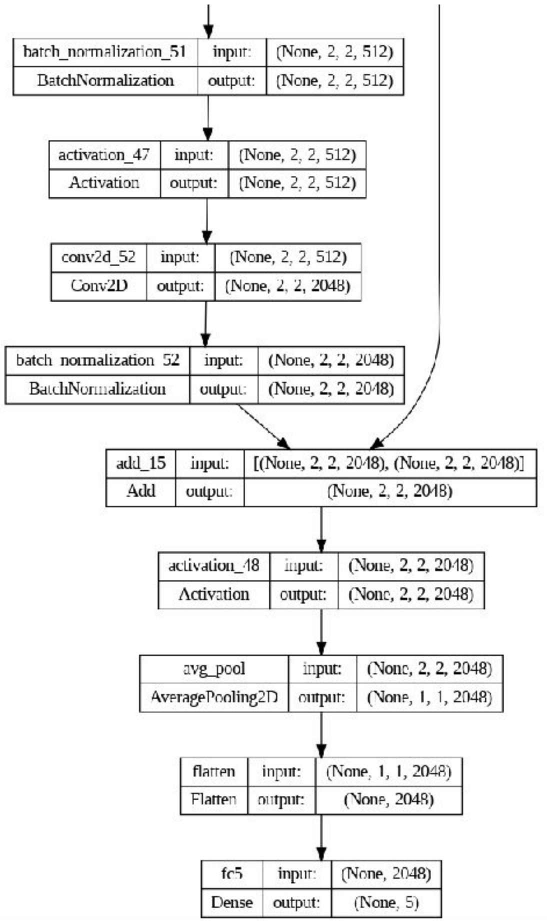
```python
def convolutional_block(X, f, filters, s=2):
    F1, F2, F3 = filters

    X_shortcut = X

    X = layers.Conv2D(F1, (1, 1), strides=(s, s), padding='valid')(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(F2, (f, f), strides=(1, 1), padding='same')(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(F3, (1, 1), strides=(1, 1), padding='valid')(X)
    X = layers.BatchNormalization(axis=3)(X)

    X_shortcut = layers.Conv2D(F3, (1, 1), strides=(s, s), padding='valid')(X_shortcut)
    X_shortcut = layers.BatchNormalization(axis=3)(X_shortcut)

    X = layers.add([X, X_shortcut])
    X = layers.Activation('relu')(X)

    return X
```

```python
def ResNet50(input_shape=(64, 64, 3), classes=5):
    X_input = layers.Input(input_shape)

    X = layers.ZeroPadding2D((3, 3))(X_input)

    X = layers.Conv2D(64, (7, 7), strides=(2, 2))(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)
    X = layers.MaxPooling2D((3, 3), strides=(2, 2))(X)

    X = convolutional_block(X, f=3, filters=[64, 64, 256], s=1)
    X = identity_block(X, 3, [64, 64, 256])
    X = identity_block(X, 3, [64, 64, 256])

    X = convolutional_block(X, f=3, filters=[128, 128, 512], s=2)
    X = identity_block(X, 3, [128, 128, 512])
    X = identity_block(X, 3, [128, 128, 512])
    X = identity_block(X, 3, [128, 128, 512])

    X = convolutional_block(X, f=3, filters=[256, 256, 1024], s=2)
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])

    X = convolutional_block(X, f=3, filters=[512, 512, 2048], s=2)
    X = identity_block(X, 3, [512, 512, 2048])
    X = identity_block(X, 3, [512, 512, 2048])
```

```python
    X = layers.AveragePooling2D((2, 2), name='avg_pool')(X)

    X = layers.Flatten()(X)

    X = layers.Dense(classes, activation='softmax', name='fc' + str(classes))(X)

    model = Model(inputs=X_input, outputs=X, name='ResNet50')

    return model
```

```python
model = Sequential()
model = ResNet50(input_shape= (64,64,3), classes=num_classes)
model.summary()
```

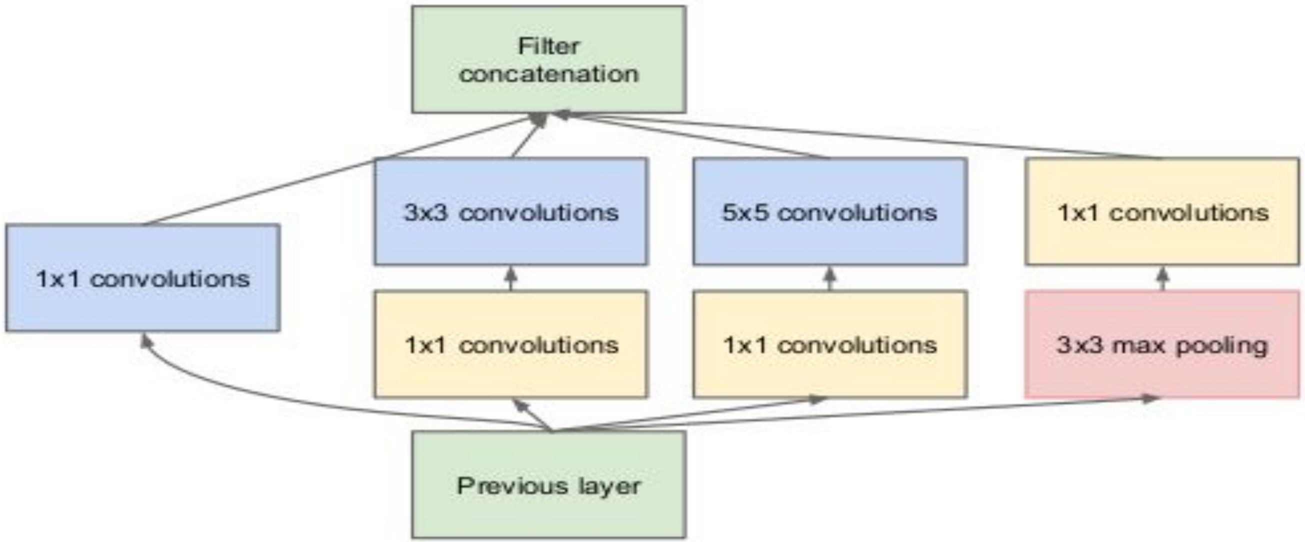
Structure of Resnet-50



Inception Net

```
def inception_module(x, f1, f2, f3):
  # 1x1 conv
  conv1 =  keras.layers.Conv2D(f1, (1,1), padding='same', activation='relu')(x)
  # 3x3 conv
  conv3 = keras.layers.Conv2D(f2, (3,3), padding='same', activation='relu')(x)
  # 5x5 conv
  conv5 = keras.layers.Conv2D(f3, (5,5), padding='same', activation='relu')(x)
  # 3x3 max pooling
  pool = keras.layers.MaxPooling2D((3,3), strides=(1,1), padding='same')(x)
  # concatenate filters
  out = keras.layers.merge.concatenate([conv1, conv3, conv5, pool])
  return out
```

Structure of Inception Net

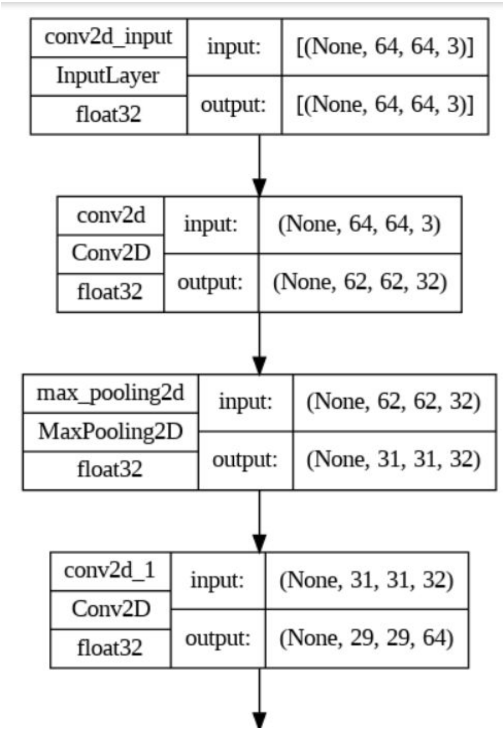


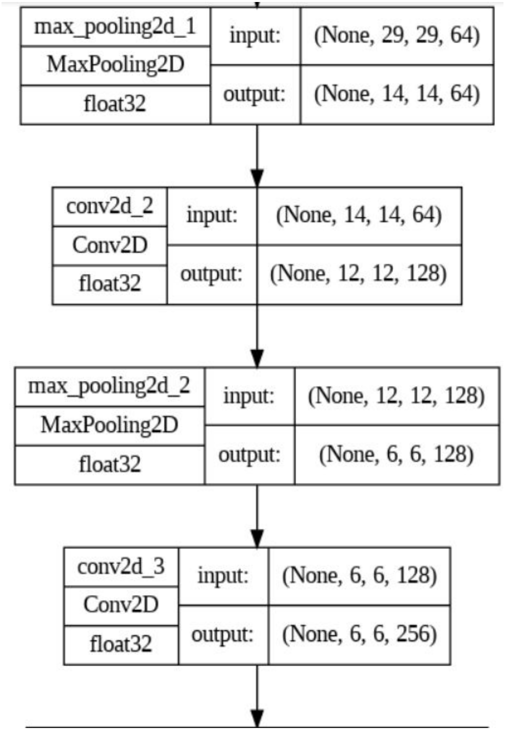
Inception Layer

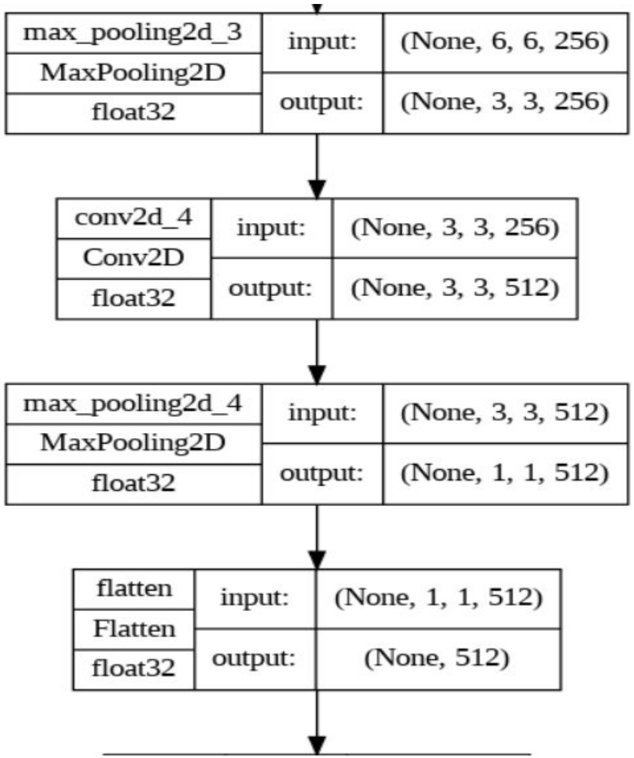
Custom CNN

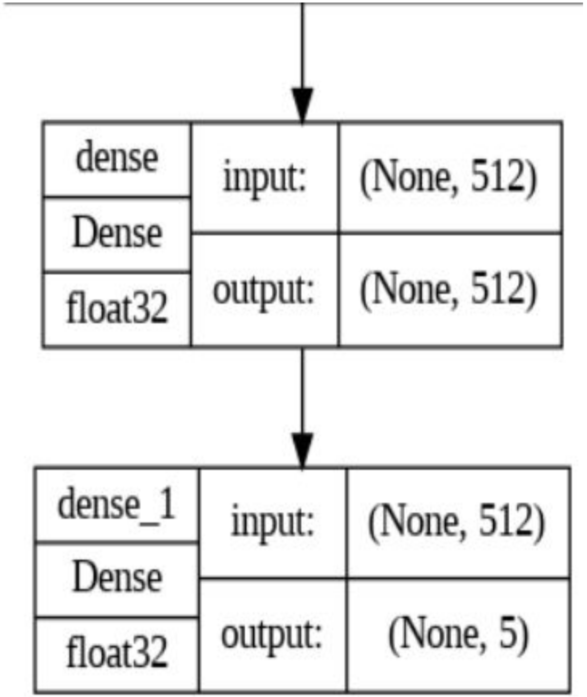
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(256, (3, 3), activation='relu',  padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(512, (3, 3), activation='relu',  padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))

model.add(layers.Dense(5, activation='softmax'))

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

history = model.fit(x_train, y_train_one_hot, epochs=10, validation_data=(x_test, y_test_one_hot))
```

Structure of Custom CNN






max_pooling2d_3
MaxPooling2D
float32
input:
(None, 6, 6, 256)
output:
(None, 3, 3, 256)
conv2d_4
Conv2D
float32
input:
(None, 3, 3, 256)
output:
(None, 3, 3, 512)
max_pooling2d_4
MaxPooling2D
float32
input:
(None, 3, 3, 512)
output:
(None, 1, 1, 512)
flatten
Flatten
float32
input:
(None, 1, 1, 512)
output:
(None, 512)


dense
Dense
float32
input:
(None, 512)
output:
(None, 512)
dense_1
Dense
float32
input:
(None, 512)
output:
(None, 5)

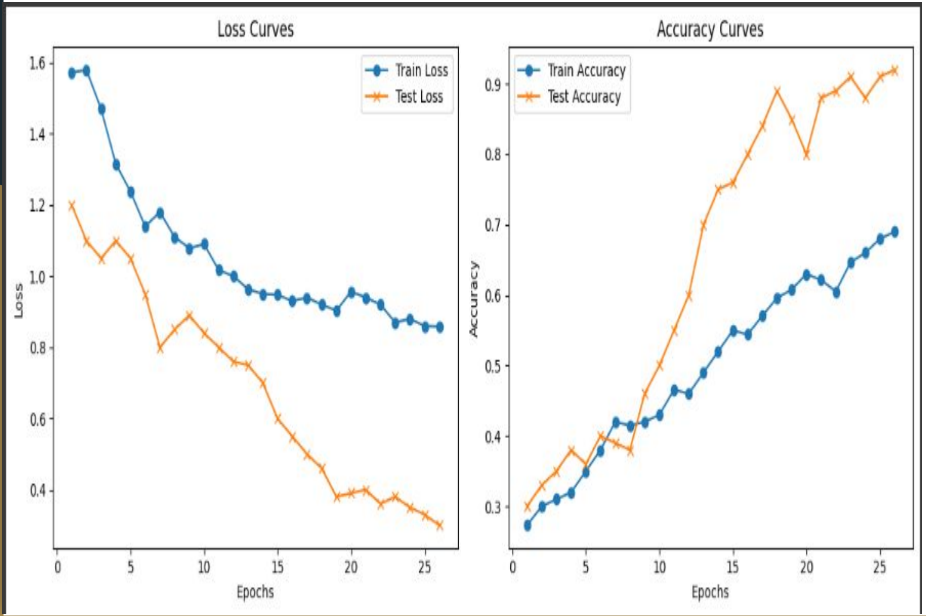
Better Model

The Resnet-50 model is the better model compared to the Inception Net model in this case.

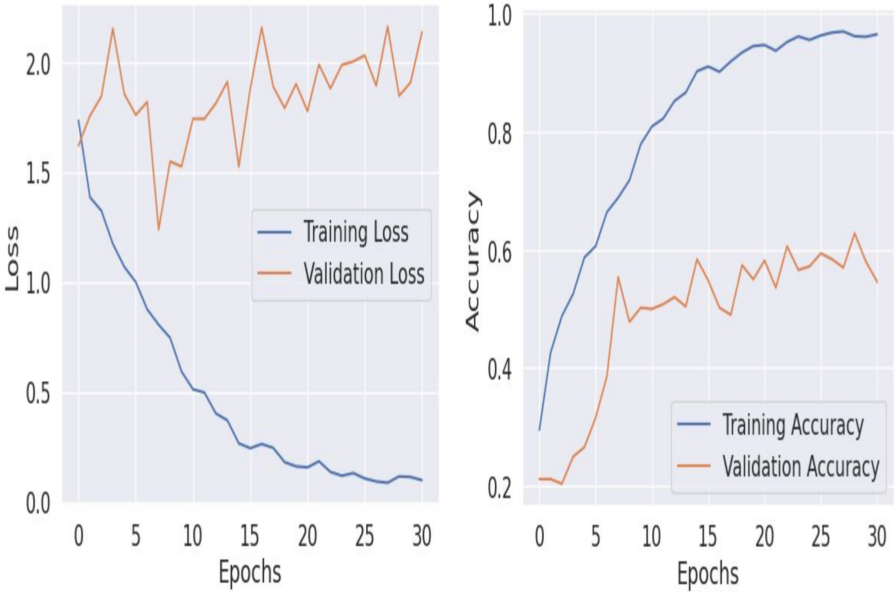
- Resnet-50 accuracy 87.74% and Inception Net accuracy 67%.
- Easier to train due to less complex architecture.
- Consistency and more deeper layers.

Train and Test Curve

Inception Model



Resnet-50 Model



Misclassified Test Images


```python
y_pred = model.predict(x_test)
y_pred_classes = y_pred.argmax(axis=1)
misclassified_indices = []
for i, (true_label, predicted_label) in enumerate(zip(y_test, y_pred_classes)):
    if true_label != predicted_label:
        misclassified_indices.append(i)
random_indices = random.sample(misclassified_indices, min(20, len(misclassified_indices)))
fig, axes = plt.subplots(3, 5, figsize=(15, 8))
for i, idx in enumerate(random_indices):
    if i >= 15:
        break
    image = X_test[idx].reshape(64, 64, 3)
    true_label = y_test[idx]
    predicted_label = y_pred_classes[idx]
    true_class_name = class_names[true_label]
    predicted_class_name = class_names[predicted_label]
    ax = axes[i // 5, i % 5]
    ax.imshow(image)
    ax.set_title(f"True: {true_class_name}\nPredicted: {predicted_class_name}", color='red')
    ax.axis('off')
plt.tight_layout()
plt.show()
```

Updated Milestone-1

Aabrar Islam - 20101361

Ayen Aziza Haque - 20301487

Simin Waliza - 20101401

Zahin Shabab - 20101165

Shahriar Azad Frahim- 20101223

Dataset

- Size of the dataset is 2414
- Test set = 250
- Train set = 2164

- Separate batch for test set
- Separate batch for train set
- Validation part has to be done manually

Classes in the dataset

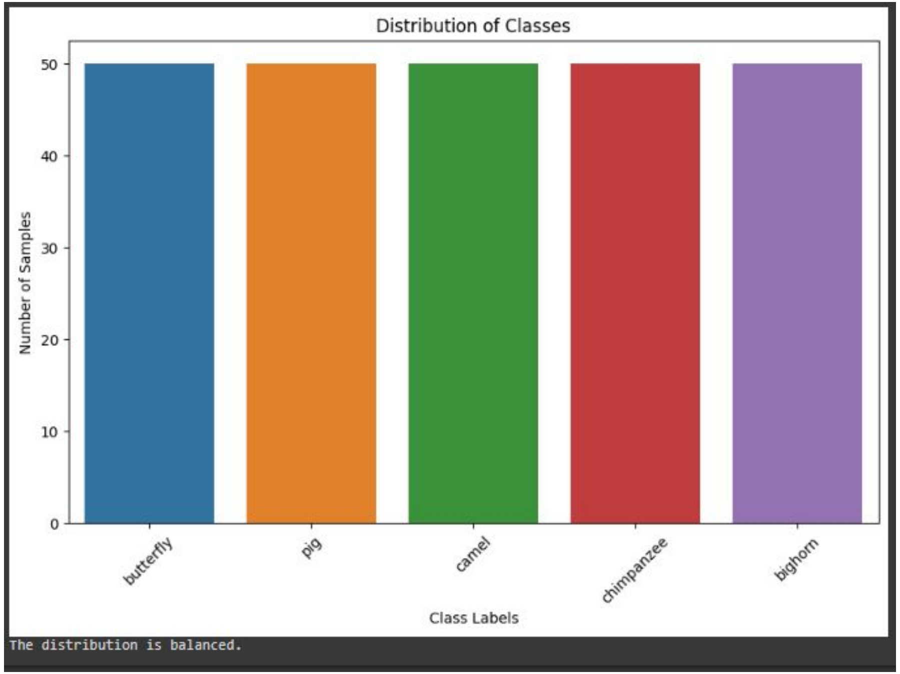
5 classes for both test and train sets are:

1. Bighorn
2. Butterfly
3. Camel
4. Chimpanzee
5. Pig.

```
Found 2164 images belonging to 5 classes.
Found 250 images belonging to 5 classes.
Class names: ['bighorn', 'butterfly', 'camel', 'chimpanzee', 'pig']
```

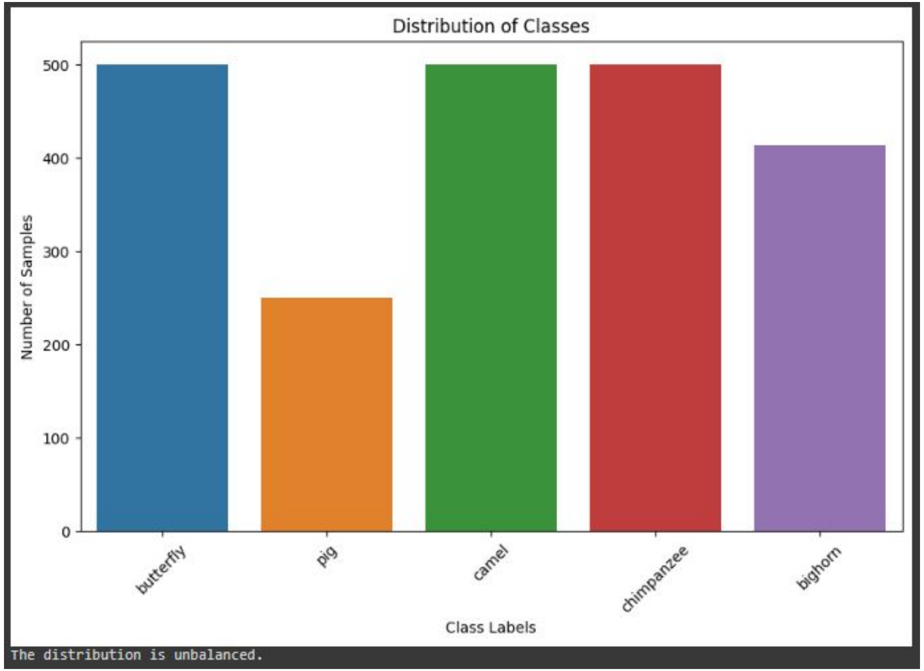
Distribution

- 5 classes
- Test(250)
  - Bighorn=50
  - Butterfly=50
  - Camel=50
  - Chimpanzee=50
  - Pig=50



Distribution

- 5 classes
- Train(2164)
  - Bighorn=414
  - Butterfly=500
  - Camel=500
  - Chimpanzee=500
  - Pig=250



Issues when dataset is unbalanced

1. **Biased Models:** Model may become biased toward the majority class, as it has more examples to learn from. This can result in poor performance on the minority class.
2. **Poor Generalization:** Unbalanced datasets can lead to poor generalization to new, unseen data, especially for the minority class. The model may simply predict the majority class most of the time, as it's more likely to be correct according to the training data distribution.
3. **Misleading Evaluation Metrics:** Common evaluation metrics like accuracy can be misleading when dealing with unbalanced datasets. A model that predicts the majority class all the time might have a high accuracy, but it's not useful if the minority class is of interest.

Tackling Unbalanced Dataset

1. **Resampling:**
    a. Oversampling: By producing duplicate or fake samples, increase the number of examples in the minority class.
    b. Undersampling: Reduce the number of instances in the majority class by deleting samples at random.
2. **Data Augmentation:** For image or text data, you can create new training examples by applying transformations or adding noise to existing samples. This can help balance the dataset.
3. **Synthetic Data Generation:** Use techniques like Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic examples for the minority class, based on the existing minority samples.

4. **Cost-sensitive Learning:** Assign different misclassification costs to different classes, emphasizing the importance of correctly classifying the minority class.
5. **Algorithm Selection:** Select techniques that are less sensitive to class imbalances, such as gradient boosting or support vector machines, which may be adjusted to place greater weight on minority class data.
6. **Change the Decision Threshold:** Adjust the categorization threshold to strike a balance between recall and precision based on your particular scenario. If the cost of false positives and false negatives differs, this can be especially helpful.
7. **Collect More Data:** In some cases, collecting more data for the minority class can be a solution if it's feasible.

Technique used to tackle unbalanced dataset

We chose the algorithm selection method to avoid or tackle the unbalanced dataset. Tree based algorithms are also performed in these scenarios and can also be boosted for unbalanced dataset

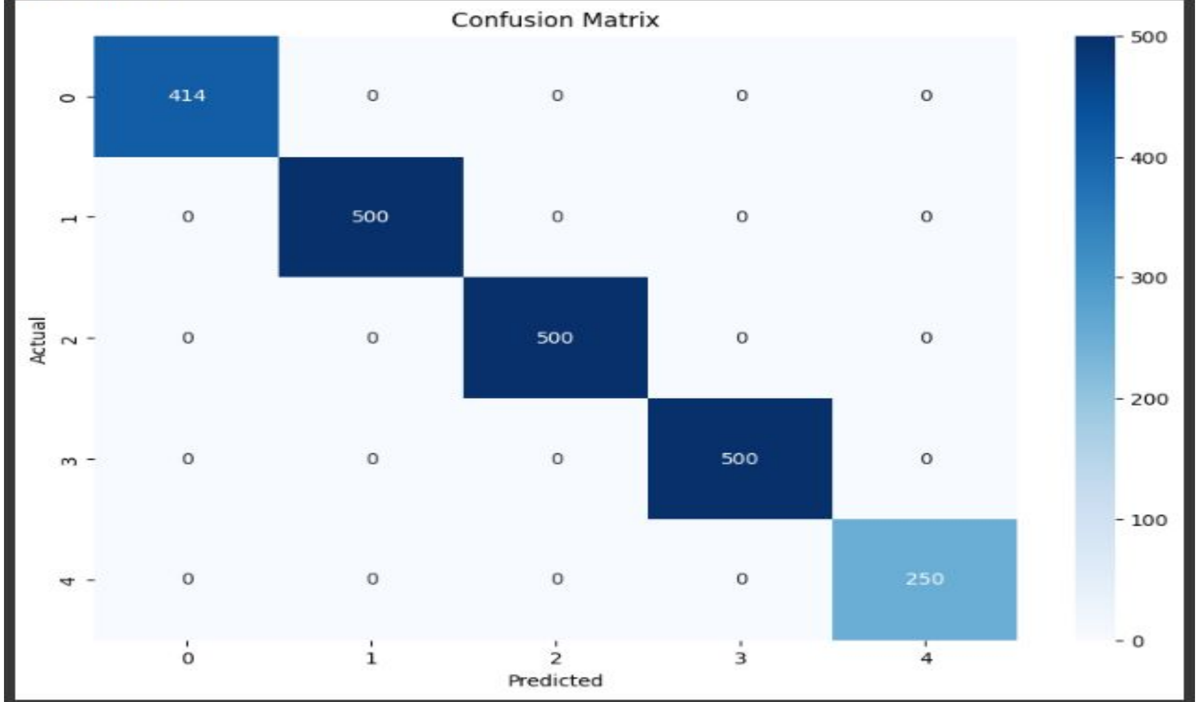
Algorithm Selection: Select techniques that are less sensitive to class imbalances, such as gradient boosting or support vector machines, which may be adjusted to place greater weight on minority class data.

Evaluation function

```
def evaluate(model, X, y):
    y_pred = model.predict(X)
    if len(y_pred.shape) > 1 and y_pred.shape[1] > 1:
        y_pred = y_pred.argmax(axis=1)
    accuracy = accuracy_score(y, y_pred)
    conf_matrix = confusion_matrix(y, y_pred)
    f1 = f1_score(y, y_pred, average='weighted')
    sensitivity, specificity, PPV, NPV = [], [], [], []
    for i in range(len(conf_matrix)):
        TP = conf_matrix[i, i]
        FN = conf_matrix[i, :].sum() - TP
        FP = conf_matrix[:, i].sum() - TP
        TN = conf_matrix.sum() - (TP + FN + FP)

        sensitivity.append(TP / (TP + FN) if (TP + FN) != 0 else 0)
        specificity.append(TN / (TN + FP) if (TN + FP) != 0 else 0)
        PPV.append(TP / (TP + FP) if (TP + FP) != 0 else 0)
        NPV.append(TN / (TN + FN) if (TN + FN) != 0 else 0)

    print("Accuracy:", accuracy)
    print("Sensitivity (per class):", sensitivity)
    print("Specificity (per class):", specificity)
    print("Positive Predictive Value (PPV) (per class):", PPV)
    print("Negative Predictive Value (NPV) (per class):", NPV)
    print("F1 Score:", f1)
    plot_confusion_matrix(conf_matrix)
```

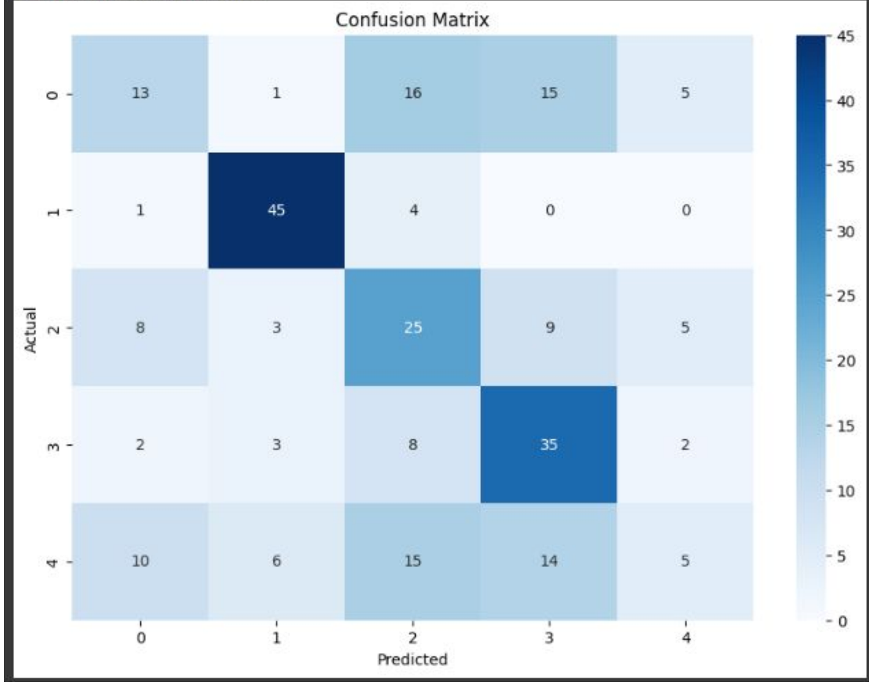
Logistic Regression(Train)

```
Accuracy: 1.0
Sensitivity (per class): [1.0, 1.0, 1.0, 1.0, 1.0]
Specificity (per class): [1.0, 1.0, 1.0, 1.0, 1.0]
Positive Predictive Value (PPV) (per class): [1.0, 1.0, 1.0, 1.0, 1.0]
Negative Predictive Value (NPV) (per class): [1.0, 1.0, 1.0, 1.0, 1.0]
F1 Score: 1.0
```



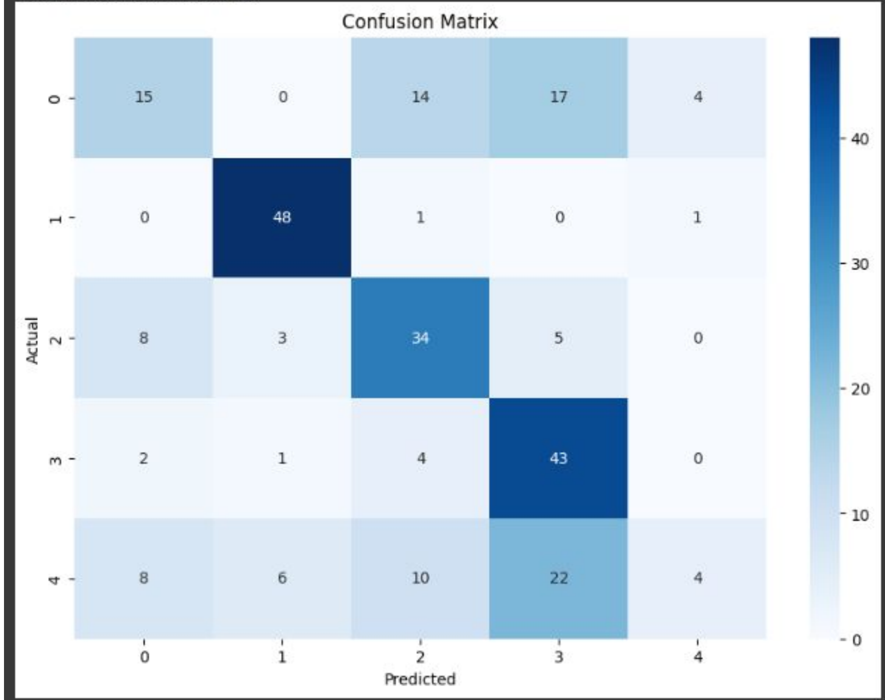
Logistic Regression(Test)

```
Accuracy: 0.492
Sensitivity (per class): [0.26, 0.9, 0.5, 0.7, 0.1]
Specificity (per class): [0.895, 0.935, 0.785, 0.81, 0.94]
Positive Predictive Value (PPV) (per class): [0.38235294117647056, 0.7758620689655172, 0.36764705882352944, 0.4794520547945205, 0.29411764705882354]
Negative Predictive Value (NPV) (per class): [0.8287037037037037, 0.9739583333333334, 0.8626373626373627, 0.9152542372881356, 0.8068669527896996]
F1 Score: 0.45698907576333186
```



Neural Network

```
8/8 [==============================] - 0s 4ms/step
Accuracy: 0.576
Sensitivity (per class): [0.3, 0.96, 0.68, 0.86, 0.08]
Specificity (per class): [0.91, 0.95, 0.855, 0.78, 0.975]
Positive Predictive Value (PPV) (per class): [0.45454545454545453, 0.8275862068965517, 0.5396825396825397, 0.4942528735632184, 0.4444444444444444]
Negative Predictive Value (NPV) (per class): [0.8387096774193549, 0.9895833333333334, 0.9144385026737968, 0.9570552147239264, 0.8091286307053942]
F1 Score: 0.5230870060284397
```



Comparison between logistic regression and neural network

- Neural network is better than logistic regression in this case.