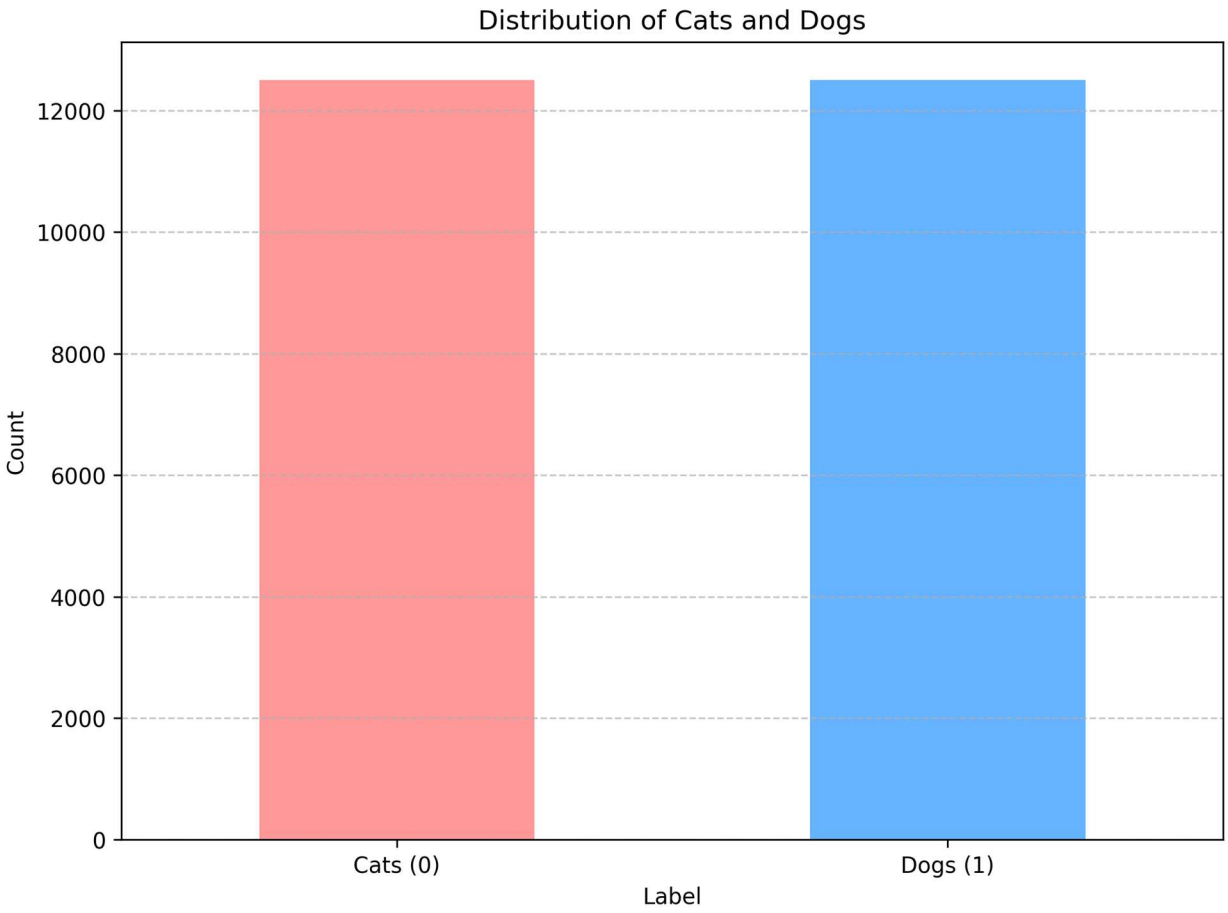
Golden Owl AI Intern Report

Exercise 1

Dataset : Cat & Dog images for Classification

Data preprocessing:

Data distribution



To add variety to existing datasets, we apply some data augmentation methods such as flipping, resizing, rotating:

```
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

I also normalize images to ensure that the pixels' value is centered around 0, which makes training process faster and more efficient.

The data will be split into 3 sets: Training set, Validation set, Testing set with ratio **7:2:1**. The validation set will be used for fine-tuning the model, testing and training set will be used for training and evaluating the model.

Building model:

I have evaluated two models: a simple CNN from scratch and transfer learning from VGG19.

Simple CNN:

```
class SimpleCNN(torch.nn.Module):
    def __init__(self, num_classes=2):
        super(SimpleCNN, self).__init__()
        self.conv1 = torch.nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = torch.nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.pool = torch.nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = torch.nn.Linear(32 * 56 * 56, 128)
        self.fc2 = torch.nn.Linear(128, num_classes)
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        x = self.pool(torch.nn.functional.relu(self.conv1(x)))
        x = self.pool(torch.nn.functional.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = torch.nn.functional.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

The neural network consists of **4 trainable** layers: **2 Convolutional layers** and **2 Fully Connected layers** with **MaxPolling** and **Dropout**.

VGG19:

With VGG19, I freeze the network to use it as a feature extraction then I modify the Fully Connected layers to adapt to the problems.

```python
def get_vgg19_model(num_classes=2, freeze_features=True):
    # Load pre-trained VGG19
    model = models.vgg19(weights=models.VGG19_Weights.IMAGENET1K_V1)

    # Freeze feature extraction layers
    if freeze_features:
        for param in model.features.parameters():
            param.requires_grad = False

    model.classifier[6] = nn.Linear(4096, num_classes)

    return model
```

Evaluation

Two models are trained using **GPU T4 x2** on Kaggle with **10 epochs** each.

To evaluate the model, we use 4 metrics: **Accuracy, Precision, Recall, and F1**

Accuracy is calculated by dividing the number of correct predictions by the number of predictions.

Precision, **Recall, F1** are mostly used when dealing with imbalanced datasets or we want to value one class more than other e.g., AIDS prediction, Crime prediction on videos.

Since the dataset is balanced and two classes are cat and dog which are like each other so I think we should focus on the **Accuracy** to evaluate the model.

Evaluation Result on testing set

	VGG19	Simple CNN
Accuracy	0.985	0.817
Precision	0.980	0.836
Recall	0.990	0.788
F1	0.985	0.811

We can see that VGG19 outperforms a Simple CNN by a large margin, this result is not surprising since VGG19 has significantly more layers than Simple CNN so that it can learn more features from an image and also the pretrained network VGG19 has been trained with ImageNet dataset which is a gigantic dataset so it can learn more about the differences between cats and dogs

Conclusion

- The **Simple CNN** still performs decently without using much storage while also has a lot less parameters.
- The **VGG19** pretrained model is excellent for these problems as a feature extraction.
- If I have more time on this task, I can do more things like **fine-tuning** more parameters such as **numbers of layers, size of kernels**, …
- The training time is quite short (about **3 minutes** each epoch)