Introduction

I used the `image_dataset_from_directory` utility to generate the datasets, and I used Keras image preprocessing layers for image standardization and data augmentation.

Setup

```
import os
import numpy as np
import keras
from keras import layers
from tensorflow import data as tf_data
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import load_img
```

Load the data: the Cats vs Dogs dataset

Raw data download

First, let's download the ZIP archive of the raw data:

```
!curl -O https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  786M  100  786M    0     0   251M      0  0:00:03  0:00:03 --:--:--  251M
```

```
!unzip -q kagglecatsanddogs_5340.zip
!ls
```

```
CDLA-Permissive-2.0.pdf   kagglecatsanddogs_5340.zip   PetImages  'readme[1].txt'   sa
```

Now we have a `PetImages` folder which contain two subfolders, `Cat` and `Dog`. Each subfolder contains image files for each category.

```
!ls PetImages
```

```
Cat  Dog
```

Filter out corrupted images

When working with lots of real-world image data, corrupted images are a common occurence. Let's filter out badly-encoded images that do not feature the string "JFIF" in their header.

```
num_skipped = 0
for folder_name in ("Cat", "Dog"):
    folder_path = os.path.join("PetImages", folder_name)
    for fname in os.listdir(folder_path):
        fpath = os.path.join(folder_path, fname)
        try:
            fobj = open(fpath, "rb")
            is_jfif = b"JFIF" in fobj.peek(10)
        finally:
            fobj.close()

        if not is_jfif:
            num_skipped += 1
            # Delete corrupted image
            os.remove(fpath)

print(f"Deleted {num_skipped} images.")
```

```
Deleted 1590 images.
```

Generate a `Dataset`

```
image_size = (180, 180)
batch_size = 128

train_ds, val_ds = keras.utils.image_dataset_from_directory(
    "PetImages",
    validation_split=0.2,
    subset="both",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 23410 files belonging to 2 classes.
Using 18728 files for training.
Using 4682 files for validation.
```

Visualize the data

Here are the first 9 images in the training dataset.

```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(np.array(images[i]).astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```

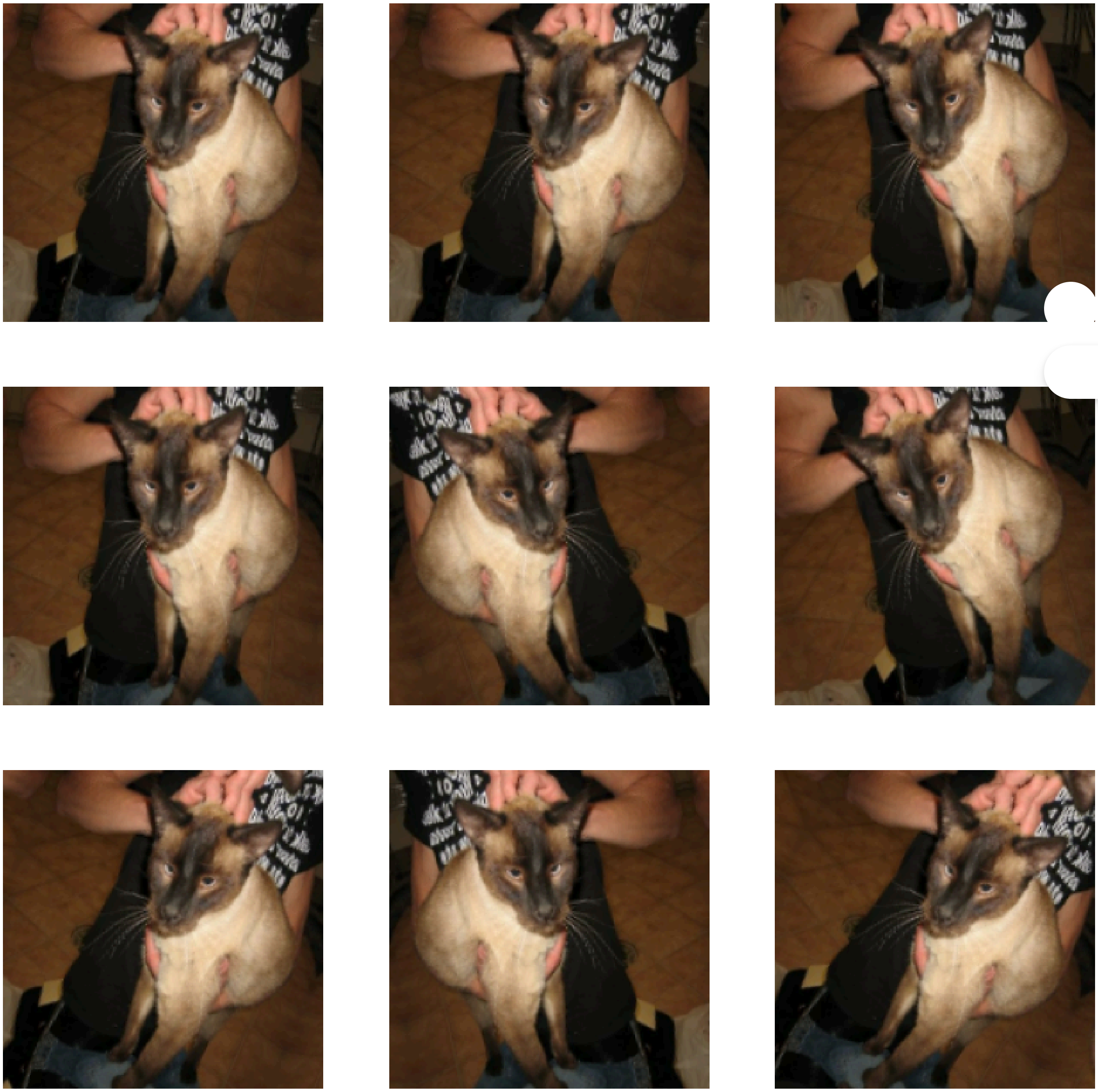
Using image data augmentation

```
data_augmentation_layers = [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
]


def data_augmentation(images):
    for layer in data_augmentation_layers:
        images = layer(images)
    return images



plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(np.array(augmented_images[0]).astype("uint8"))
        plt.axis("off")
```



Configure the dataset for performance

```
train_ds = train_ds.map(
    lambda img, label: (data_augmentation(img), label),
    num_parallel_calls=tf_data.AUTOTUNE,
)

train_ds = train_ds.prefetch(tf_data.AUTOTUNE)
val_ds = val_ds.prefetch(tf_data.AUTOTUNE)
```

Build a model

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)


    x = layers.Rescaling(1.0 / 255)(inputs)
    x = layers.Conv2D(128, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x
```

Train the model

```
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
```

```
epochs = 10
model.compile(
    optimizer=keras.optimizers.Adam(3e-4),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy(name="acc")],
)
model.fit(
    train_ds,
    epochs=epochs,
    validation_data=val_ds,
)
```

```
Epoch 1/10
147/147 [==============================] - 214s 1s/step - loss: 0.4617 - acc: 0.7683 -
Epoch 2/10
147/147 [==============================] - 219s 1s/step - loss: 0.3754 - acc: 0.8228 -
Epoch 3/10
147/147 [==============================] - 209s 1s/step - loss: 0.3204 - acc: 0.8550 -
Epoch 4/10
147/147 [==============================] - 209s 1s/step - loss: 0.2710 - acc: 0.8776 -
Epoch 5/10
147/147 [==============================] - 209s 1s/step - loss: 0.2384 - acc: 0.8951 -
```