Experiment: 1

Aim

To develop and train a Convolutional Neural Network (CNN) for accurate classification of images into two distinct categories, utilizing data augmentation techniques and regularization to improve model generalization and reduce overfitting.

Theory

The image classification experiment aims to categorize images into two classes using a Convolutional Neural Network (CNN). The dataset comprises 8005 images, split into 6404 for training and 1601 for validation. Data augmentation techniques like random flipping, rotation, and translation are applied to improve model generalization. The CNN architecture includes convolutional layers, batch normalization, pooling, global average pooling, and a dropout layer to reduce overfitting, with a final dense layer using a sigmoid activation for binary classification. The model is trained using the Adam optimizer, binary cross-entropy loss, and binary accuracy as the evaluation metric, to achieve high classification accuracy while minimizing overfitting

In [ ]:

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

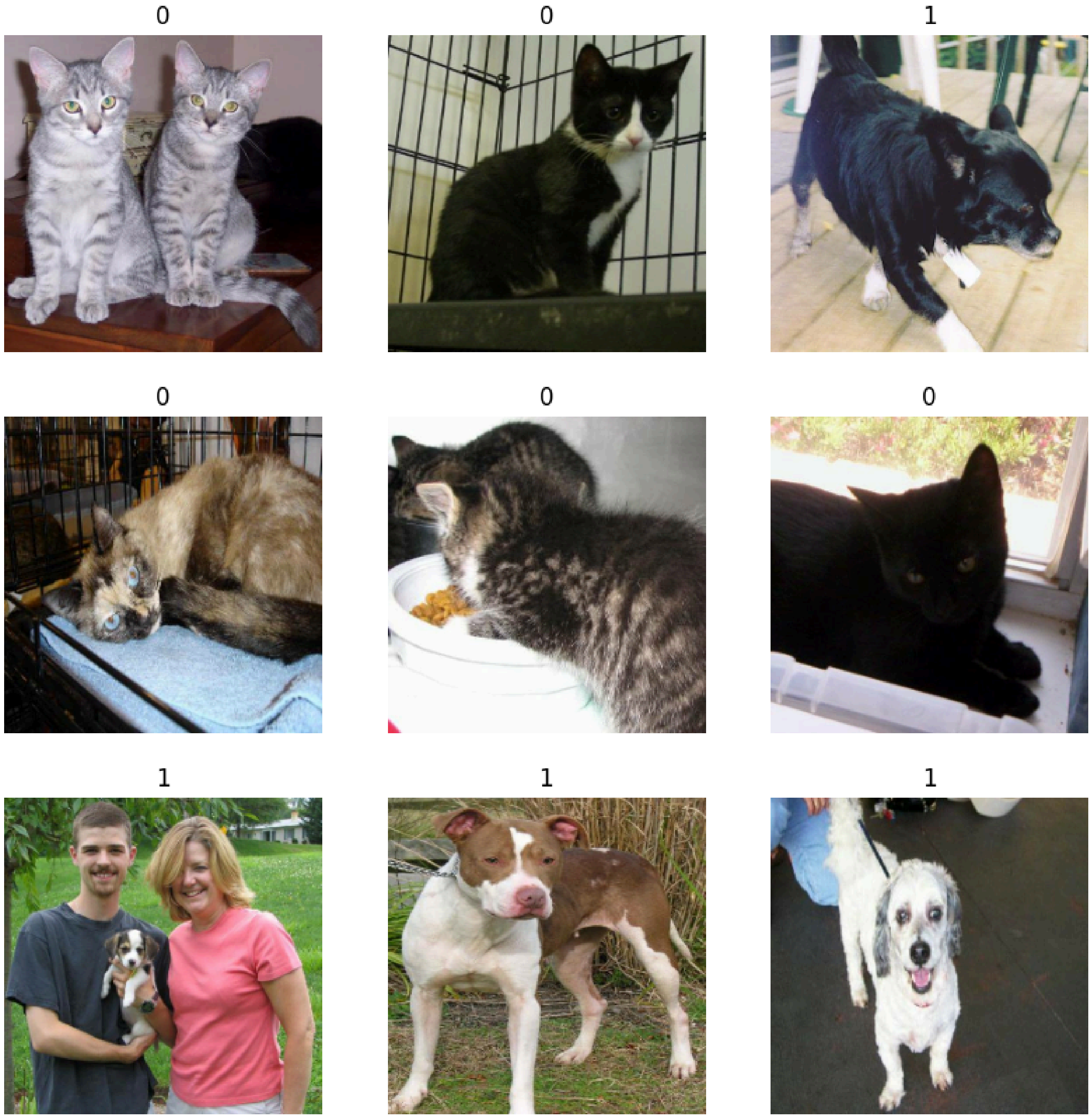
In [ ]:

```
image_size = (256,256)
batch_size = 32
img_h = 256
img_w = 256

train_ds, val_ds = keras.utils.image_dataset_from_directory(
    "/workspace/ADNN/Exp-1/Data/training_set/training_set",
    validation_split=0.2,
    subset="both",
    seed=1337,
    batch_size=batch_size,
    image_size = image_size,
)
```

```
Found 8005 files belonging to 2 classes.
Using 6404 files for training.
Using 1601 files for validation.
```

In [ ]:

```
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")
```



In [ ]:
```
data_augmentation_layers = [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomTranslation((-0.2, 0.3),(-0.2, 0.3)),
]

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

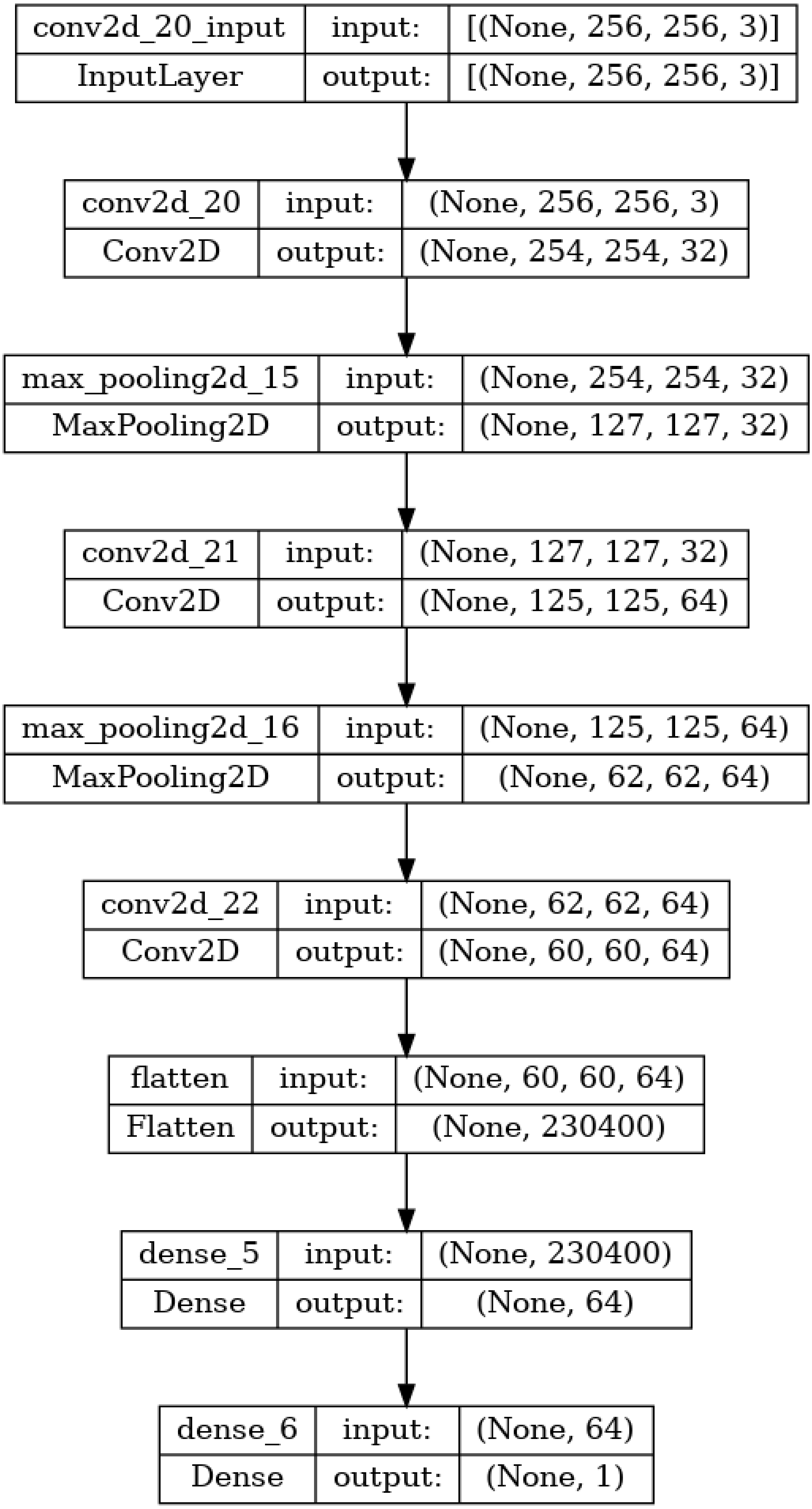
In [ ]:
```
#parallel mapping
train_ds = train_ds.map(
    lambda img, label: (data_augmentation(img), label),
    num_parallel_calls=tf_data.AUTOTUNE,
)

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

In [ ]:
```
from tensorflow.keras import layers, models, Input
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
def make_simple_model(input_shape, num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    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(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    if num_classes == 2:
        model.add(layers.Dense(1, activation='sigmoid'))
    else:
        model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Specify input shape and number of classes
input_shape = (256, 256, 3)
num_classes = 2  # Update this to your number of classes

model = make_simple_model(input_shape, num_classes)
model.summary()

# Plot model architecture
keras.utils.plot_model(model, show_shapes=True)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_20 (Conv2D)          (None, 254, 254, 32)      896       
                                                                 
 max_pooling2d_15 (MaxPooli  (None, 127, 127, 32)      0         
 ng2D)                                                           
                                                                 
 conv2d_21 (Conv2D)          (None, 125, 125, 64)      18496     
                                                                 
 max_pooling2d_16 (MaxPooli  (None, 62, 62, 64)        0         
 ng2D)                                                           
                                                                 
 conv2d_22 (Conv2D)          (None, 60, 60, 64)        36928     
                                                                 
 flatten (Flatten)           (None, 230400)            0         
                                                                 
 dense_5 (Dense)             (None, 64)                14745664  
                                                                 
 dense_6 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 14802049 (56.47 MB)
Trainable params: 14802049 (56.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Out[ ]:



```
conv2d_20_input | input:  | [(None, 256, 256, 3)]
InputLayer      | output: | [(None, 256, 256, 3)]

conv2d_20       | input:  | (None, 256, 256, 3)
Conv2D          | output: | (None, 254, 254, 32)

max_pooling2d_15 | input:  | (None, 254, 254, 32)
MaxPooling2D     | output: | (None, 127, 127, 32)

conv2d_21       | input:  | (None, 127, 127, 32)
Conv2D          | output: | (None, 125, 125, 64)

max_pooling2d_16 | input:  | (None, 125, 125, 64)
MaxPooling2D     | output: | (None, 62, 62, 64)

conv2d_22       | input:  | (None, 62, 62, 64)
Conv2D          | output: | (None, 60, 60, 64)

flatten         | input:  | (None, 60, 60, 64)
Flatten         | output: | (None, 230400)

dense_5         | input:  | (None, 230400)
Dense           | output: | (None, 64)

dense_6         | input:  | (None, 64)
Dense           | output: | (None, 1)
```

In [ ]:

```
epochs = 10

# callbacks = [
#     keras.callbacks.ModelCheckpoint("save_at_{epoch}.keras"),
# ]
model.compile(
    optimizer=keras.optimizers.Adam(3e-4),
    loss= keras.losses.BinaryCrossentropy(),
    metrics=[keras.metrics.BinaryAccuracy(name="acc")],
)
model.fit(
    train_ds,
    epochs=epochs,
#     callbacks=callbacks,""
    validation_data=val_ds,
)
```

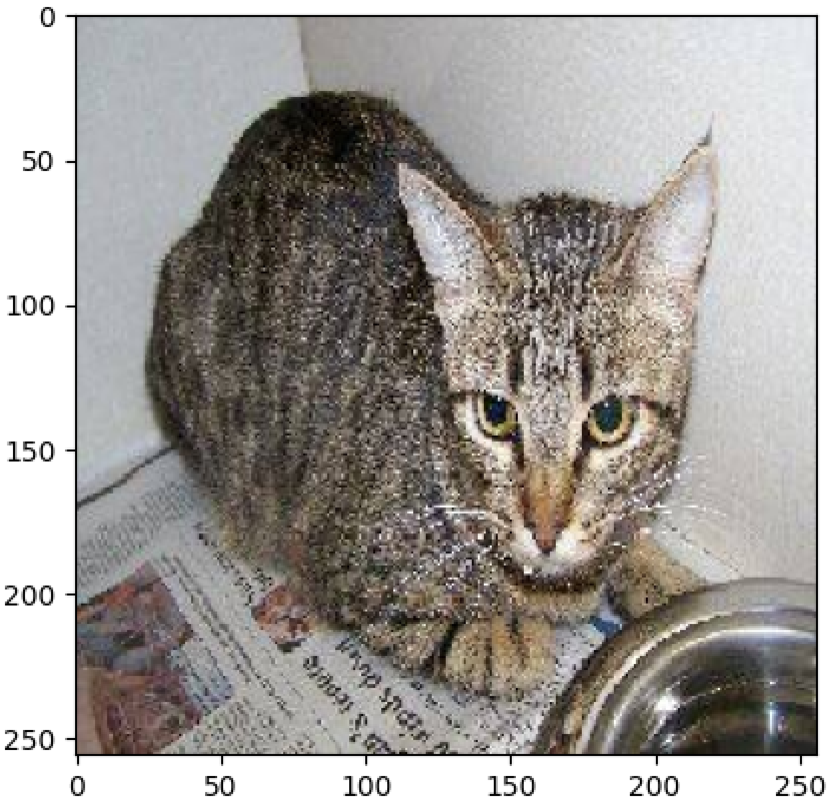
```
Epoch 1/10
201/201 [==============================] - 13s 38ms/step - loss: 11.7637 - acc: 0.5314 - val_
loss: 0.6909 - val_acc: 0.5728
Epoch 2/10
201/201 [==============================] - 6s 30ms/step - loss: 0.6878 - acc: 0.5531 - val_lo
ss: 0.6967 - val_acc: 0.5315
Epoch 3/10
201/201 [==============================] - 6s 29ms/step - loss: 0.6880 - acc: 0.5428 - val_lo
ss: 0.6900 - val_acc: 0.5778
Epoch 4/10
201/201 [==============================] - 6s 29ms/step - loss: 0.6899 - acc: 0.5515 - val_lo
ss: 0.6875 - val_acc: 0.5709
Epoch 5/10
201/201 [==============================] - 6s 29ms/step - loss: 0.6886 - acc: 0.5434 - val_lo
ss: 0.6838 - val_acc: 0.5659
Epoch 6/10
201/201 [==============================] - 6s 29ms/step - loss: 0.6862 - acc: 0.5504 - val_lo
ss: 0.6969 - val_acc: 0.5209
Epoch 7/10
201/201 [==============================] - 6s 30ms/step - loss: 0.6879 - acc: 0.5493 - val_lo
ss: 0.6921 - val_acc: 0.5378
Epoch 8/10
201/201 [==============================] - 6s 29ms/step - loss: 0.6915 - acc: 0.5309 - val_lo
ss: 0.6851 - val_acc: 0.5497
Epoch 9/10
201/201 [==============================] - 6s 30ms/step - loss: 0.6880 - acc: 0.5373 - val_lo
ss: 0.6895 - val_acc: 0.5540
Epoch 10/10
201/201 [==============================] - 6s 28ms/step - loss: 0.6872 - acc: 0.5540 - val_lo
ss: 0.6842 - val_acc: 0.5522
```

Out[ ]:

```
<keras.src.callbacks.History at 0x7fa7e05fe350>
```

In [ ]:

```
import tensorflow as tf
img = keras.utils.load_img("/workspace/ADNN/Exp-1/Data/test_set/test_set/cats/cat.5000.jpg",
plt.imshow(img)
plt.show()

# Convert the image to an array and preprocess it
img_array = keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create a batch axis

# Make predictions
predictions = model.predict(img_array)
score = tf.nn.sigmoid(predictions[0][0])
print(f"This image is {100 * (1 - score):.2f}% cat and {100 * score:.2f}% dog.")
```



```
1/1 [==============================] - 0s 84ms/step
This image is 37.72% cat and 62.28% dog.
```

In [ ]: