```
# Tải và giải nén dataset từ Dropbox
#test_set
!wget -O dataset.zip "https://www.dropbox.com/scl/fo/sdxxm2k4j4utqbfj3h1a3/AH8W1_ox_tOSiwWkjTbcJvU?rlkey=5611d1leoelid043o9jofruzc&st=ocir78
!unzip -q dataset.zip -d /content/test_set
```

Hiện kết quả đã ẩn

```
#down trainset
!wget -O trainset.zip "https://www.dropbox.com/scl/fo/bxmrajz6heawmgby2ff2v/AF01_DtCXgIzexXNxBIcXpo?rlkey=01w6b259di2ylttmccahptbvv&st=i3czh
!unzip -q trainset.zip -d /content/train_set
```

Hiện kết quả đã ẩn

```
# get dataset from folder test_set and training_set
import os

train_dir = 'training_set'
test_dir = 'test_set'


# Kiểm tra cấu trúc thư mục
print("Cấu trúc thư mục training_set:")
for root, dirs, files in os.walk(train_dir):
    level = root.replace(train_dir, '').count(os.sep)
    indent = ' ' * 2 * level
    print(f'{indent}{os.path.basename(root)}/')
    subindent = ' ' * 2 * (level + 1)
    for file in files[:5]:  # Chỉ hiển thị 5 file đầu
        print(f'{subindent}{file}')
    if len(files) > 5:
        print(f'{subindent}... và {len(files)-5} file khác')
```

```
Cấu trúc thư mục training_set:
train_set/
  dogs/
    dog.1314.jpg
    dog.3802.jpg
    dog.3877.jpg
    dog.1890.jpg
    dog.2437.jpg
    ... và 3995 file khác
  cats/
    cat.1987.jpg
    cat.3339.jpg
    cat.1601.jpg
    cat.3057.jpg
    cat.1386.jpg
    ... và 3995 file khác
```

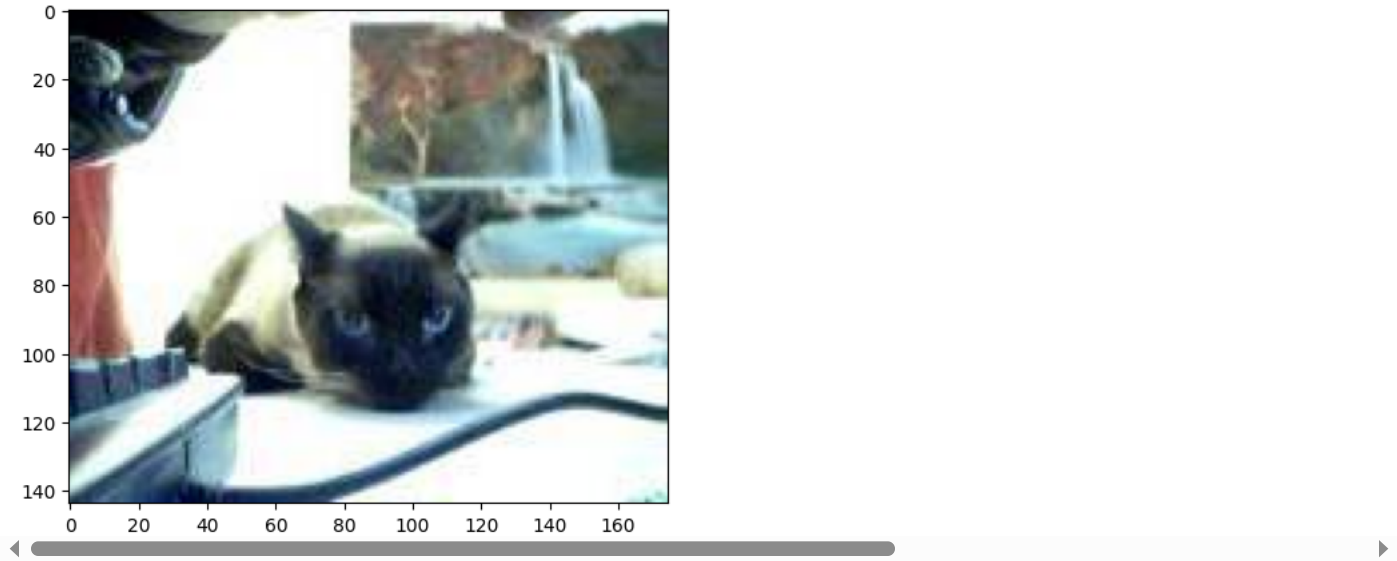
```
print("\nCấu trúc thư mục test_set:")
for root, dirs, files in os.walk(test_dir):
    level = root.replace(test_dir, '').count(os.sep)
    indent = ' ' * 2 * level
    print(f'{indent}{os.path.basename(root)}/')
    subindent = ' ' * 2 * (level + 1)
    for file in files[:5]:
        print(f'{subindent}{file}')
    if len(files) > 5:
        print(f'{subindent}... và {len(files)-5} file khác')
```

```

Cấu trúc thư mục test_set:
test_set/
  dogs/
    dog.4403.jpg
    dog.4927.jpg
    dog.4363.jpg
    dog.4775.jpg
    dog.4347.jpg
    ... và 995 file khác
  cats/
    cat.4831.jpg
    cat.4094.jpg
    cat.4138.jpg
    cat.4164.jpg
    cat.4959.jpg
```

```
          ... và 995 file khác
```

```
# get cat.1.jpeg from train_set/cats/cat.1.jpeg
import matplotlib.pyplot as plt
from PIL import Image
# Hiển thị một hình ảnh mẫu từ tập huấn luyện
img = 'training_set/cats/cat.5.jpg'
image = Image.open(img)
plt.imshow(image)
# hiển thị shape của hình ảnh
print({image.size})
```

```
{(175, 144)}
```




```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt


IMG_HEIGHT = 150
IMG_WIDTH = 150
BATCH_SIZE = 32


train_datagen = ImageDataGenerator(
    rescale=1./255,           # Chuẩn hóa pixel về [0,1]
    rotation_range=20,        # Xoay ảnh
    width_shift_range=0.2,    # Dịch chuyển ngang
    height_shift_range=0.2,   # Dịch chuyển dọc
    shear_range=0.2,          # Biến dạng
    zoom_range=0.2,           # Thu phóng
    horizontal_flip=True,     # Lật ngang
    fill_mode='nearest',      # Điền pixel
    validation_split=0.2      # Chia 20% cho validation
)


test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',      # Binary classification (dogs vs cats)
    subset='training'         # Sử dụng 80% cho training
)
```

```
Found 6400 images belonging to 2 classes.
```

```
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
```

```
    class_mode='binary',
    subset='validation'       # Sử dụng 20% cho validation
)
```

```
Found 1600 images belonging to 2 classes.
```

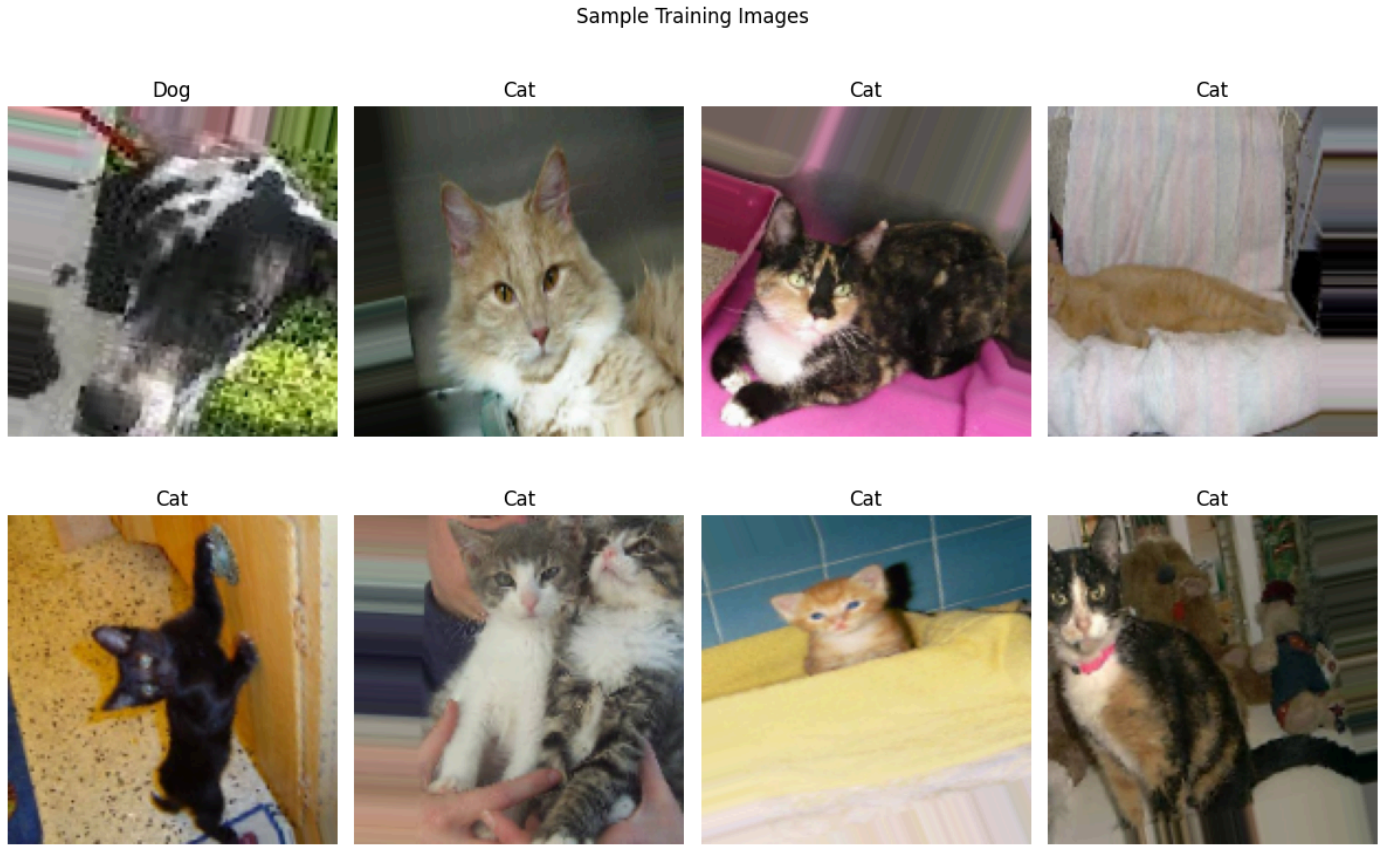
```
# Tạo test generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=False             # Không shuffle để dễ đánh giá
)
```

```
Found 2000 images belonging to 2 classes.
```

```
# In thông tin về dataset
print(f"\nThông tin dataset:")
print(f"Training samples: {train_generator.samples}")
print(f"Validation samples: {validation_generator.samples}")
print(f"Test samples: {test_generator.samples}")
print(f"Number of classes: {train_generator.num_classes}")
print(f"Class indices: {train_generator.class_indices}")
```

```

Thông tin dataset:
Training samples: 6400
Validation samples: 1600
Test samples: 2000
Number of classes: 2
Class indices: {'cats': 0, 'dogs': 1}
```

```
def plot_sample_images(generator, title):
    plt.figure(figsize=(12, 8))
    batch_images, batch_labels = next(generator)

    for i in range(min(8, len(batch_images))):
        plt.subplot(2, 4, i + 1)
        plt.imshow(batch_images[i])
        class_name = 'Dog' if batch_labels[i] == 1 else 'Cat'
        plt.title(f'{class_name}')
        plt.axis('off')

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

# Hiển thị ảnh mẫu từ training set
plot_sample_images(train_generator, 'Sample Training Images')
```




```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Xây dựng mô hình CNN
def create_cnn_model():
    model = Sequential([
        # Khối tích chập thứ nhất
        Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
        MaxPooling2D(2, 2),

        # Khối tích chập thứ hai
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        # Khối tích chập thứ ba
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        # Khối tích chập thứ tư
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        # Flatten để chuyển từ 2D sang 1D
        Flatten(),

        # Lớp Fully Connected
        Dense(512, activation='relu'),
        Dropout(0.5),  # Dropout để tránh overfitting

        # Lớp đầu ra với Sigmoid cho binary classification
        Dense(1, activation='sigmoid')  # Sigmoid cho binary classification (chó vs mèo)
    ])
```

```
    return model


# Tạo mô hình
model = create_cnn_model()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile mô hình
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',     # Binary crossentropy cho binary classification
    metrics=['accuracy']
)


# Hiển thị kiến trúc mô hình
print("Kiến trúc mô hình CNN:")
model.summary()
```

```
Kiến trúc mô hình CNN:
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

```
Total params: 3,453,121 (13.17 MB)
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

callbacks = [
    # Early stopping để tránh overfitting
    EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
    ),

    # Lưu model tốt nhất
    ModelCheckpoint(
        'best_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    ),

    # Giảm learning rate khi loss không cải thiện
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=3,
        min_lr=1e-7,
        verbose=1
```

```
    )
]


EPOCHS = 100

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks,
    verbose=1
)
```

```
Epoch 7/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 216ms/step - accuracy: 0.6979 - loss: 0.5817
Epoch 7: val_accuracy did not improve from 0.71625
200/200 ━━━━━━━━━━━━━━━━━━━━ 54s 269ms/step - accuracy: 0.6979 - loss: 0.5817 - val_accuracy: 0.6881 - val_loss: 0.5751 - learning_ra
Epoch 8/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 216ms/step - accuracy: 0.6962 - loss: 0.5703
Epoch 8: val_accuracy did not improve from 0.71625
200/200 ━━━━━━━━━━━━━━━━━━━━ 54s 268ms/step - accuracy: 0.6962 - loss: 0.5703 - val_accuracy: 0.7031 - val_loss: 0.5737 - learning_ra
Epoch 9/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step - accuracy: 0.7042 - loss: 0.5725
Epoch 9: val_accuracy improved from 0.71625 to 0.73438, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
200/200 ━━━━━━━━━━━━━━━━━━━━ 63s 317ms/step - accuracy: 0.7043 - loss: 0.5725 - val_accuracy: 0.7344 - val_loss: 0.5419 - learning_ra
Epoch 10/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step - accuracy: 0.7286 - loss: 0.5420
Epoch 10: val_accuracy did not improve from 0.73438
200/200 ━━━━━━━━━━━━━━━━━━━━ 72s 268ms/step - accuracy: 0.7286 - loss: 0.5421 - val_accuracy: 0.7275 - val_loss: 0.5390 - learning_ra
Epoch 11/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step - accuracy: 0.7330 - loss: 0.5449
Epoch 11: val_accuracy did not improve from 0.73438
200/200 ━━━━━━━━━━━━━━━━━━━━ 53s 264ms/step - accuracy: 0.7330 - loss: 0.5449 - val_accuracy: 0.7031 - val_loss: 0.5616 - learning_ra
Epoch 12/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step - accuracy: 0.7374 - loss: 0.5326
Epoch 12: val_accuracy improved from 0.73438 to 0.74000, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
200/200 ━━━━━━━━━━━━━━━━━━━━ 82s 265ms/step - accuracy: 0.7374 - loss: 0.5326 - val_accuracy: 0.7400 - val_loss: 0.5204 - learning_ra
Epoch 13/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 218ms/step - accuracy: 0.7470 - loss: 0.5232
Epoch 13: val_accuracy did not improve from 0.74000
200/200 ━━━━━━━━━━━━━━━━━━━━ 82s 264ms/step - accuracy: 0.7470 - loss: 0.5231 - val_accuracy: 0.7344 - val_loss: 0.5220 - learning_ra
Epoch 14/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step - accuracy: 0.7410 - loss: 0.5202
Epoch 14: val_accuracy improved from 0.74000 to 0.74937, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
200/200 ━━━━━━━━━━━━━━━━━━━━ 82s 263ms/step - accuracy: 0.7410 - loss: 0.5202 - val_accuracy: 0.7494 - val_loss: 0.5058 - learning_ra
Epoch 15/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step - accuracy: 0.7508 - loss: 0.5088
Epoch 15: val_accuracy did not improve from 0.74937
200/200 ━━━━━━━━━━━━━━━━━━━━ 53s 264ms/step - accuracy: 0.7508 - loss: 0.5087 - val_accuracy: 0.7281 - val_loss: 0.5563 - learning_ra
Epoch 16/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step - accuracy: 0.7595 - loss: 0.5016
Epoch 16: val_accuracy improved from 0.74937 to 0.77750, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
200/200 ━━━━━━━━━━━━━━━━━━━━ 82s 266ms/step - accuracy: 0.7595 - loss: 0.5016 - val_accuracy: 0.7775 - val_loss: 0.4661 - learning_ra
Epoch 17/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step - accuracy: 0.7688 - loss: 0.4765
Epoch 17: val_accuracy did not improve from 0.77750
200/200 ━━━━━━━━━━━━━━━━━━━━ 53s 264ms/step - accuracy: 0.7687 - loss: 0.4766 - val_accuracy: 0.7713 - val_loss: 0.4704 - learning_ra
Epoch 18/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step - accuracy: 0.7821 - loss: 0.4685
Epoch 18: val_accuracy improved from 0.77750 to 0.78750, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
200/200 ━━━━━━━━━━━━━━━━━━━━ 53s 267ms/step - accuracy: 0.7820 - loss: 0.4685 - val_accuracy: 0.7875 - val_loss: 0.4891 - learning_ra
Epoch 19/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step - accuracy: 0.7867 - loss: 0.4569
Epoch 19: val_accuracy improved from 0.78750 to 0.79188, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is co
```

```
# Vẽ biểu đồ training history
import matplotlib.pyplot as plt

def plot_training_history(history):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

    # Biểu đồ Loss
```

```
    ax1.plot(history.history['loss'], label='Training Loss')
    ax1.plot(history.history['val_loss'], label='Validation Loss')
    ax1.set_title('Model Loss')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Loss')
    ax1.legend()
    ax1.grid(True)

    # Biểu đồ Accuracy
    ax2.plot(history.history['accuracy'], label='Training Accuracy')
    ax2.plot(history.history['val_accuracy'], label='Validation Accuracy')
    ax2.set_title('Model Accuracy')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Accuracy')
    ax2.legend()
    ax2.grid(True)

    plt.tight_layout()
    plt.show()

# Vẽ biểu đồ
plot_training_history(history)
```


```
# Đánh giá mô hình trên test set
test_loss, test_accuracy = model.evaluate(test_generator, verbose=1)
print(f"\nKết quả trên Test Set:")
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Dự đoán trên test set
predictions = model.predict(test_generator)
predicted_classes = (predictions > 0.5).astype(int)
```

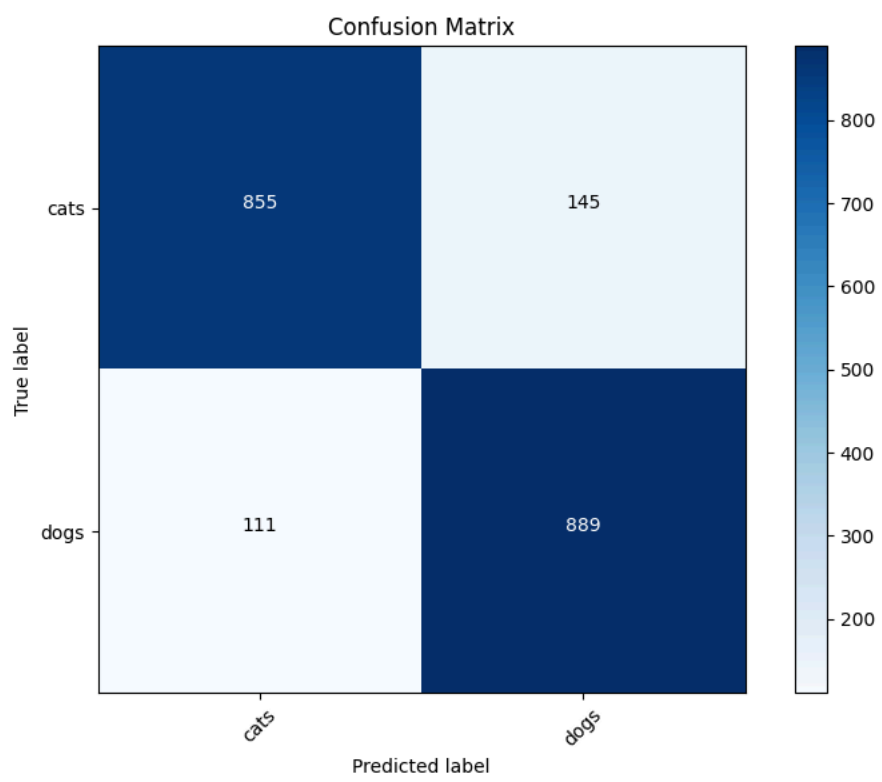
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.8704 - loss: 0.3241

Kết quả trên Test Set:
Test Loss: 0.3015
Test Accuracy: 0.8720
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step
```

```
def show_predictions(generator, predictions, num_images=8):
    plt.figure(figsize=(15, 8))

    # Reset generator
    generator.reset()
    batch_images, batch_labels = next(generator)

    for i in range(min(num_images, len(batch_images))):
        plt.subplot(2, 4, i + 1)
        plt.imshow(batch_images[i])

        actual_class = 'Dog' if batch_labels[i] == 1 else 'Cat'
        predicted_class = 'Dog' if predictions[i] > 0.5 else 'Cat'
        confidence = predictions[i][0] if predictions[i] > 0.5 else 1 - predictions[i][0]
```

```
        color = 'green' if actual_class == predicted_class else 'red'
        plt.title(f'Actual: {actual_class}\nPredicted: {predicted_class}\nConfidence: {confidence:.2f}',
                  color=color)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Hiển thị kết quả dự đoán
show_predictions(test_generator, predictions, 8)
```


```
# confusion matrix
from sklearn.metrics import confusion_matrix, classification_report
def plot_confusion_matrix(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()

    tick_marks = range(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i in range(len(classes)):
        for j in range(len(classes)):
            plt.text(j, i, cm[i, j], horizontalalignment='center',
                     color='white' if cm[i, j] > thresh else 'black')

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

```
# Lấy nhãn thực tế từ test_generator
y_true = test_generator.classes
# Lấy nhãn dự đoán từ predicted_classes
y_pred = predicted_classes.flatten()
# Lấy tên các lớp từ class_indices
class_names = list(test_generator.class_indices.keys())
# Vẽ confusion matrix
plot_confusion_matrix(y_true, y_pred, class_names)
```




```
#classification report
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=class_names))
```

```
Classification Report:
              precision    recall  f1-score   support

        cats       0.89      0.85      0.87      1000
        dogs       0.86      0.89      0.87      1000

    accuracy                           0.87      2000
   macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000
```

```
dog_image = 'dog_sample.jfif'
not_dog_image = 'not_dog_sample.jfif'

plt.imshow(Image.open(dog_image))
plt.axis('off')
plt.show()
```

```
plt.imshow(Image.open(not_dog_image))
plt.axis('off')
plt.show()
```

```
from PIL import Image
from tensorflow.keras.preprocessing.image import load_img, img_to_array


def predict_single_image(model, img_path, img_size=(150, 150)):
    """
    Dự đoán một hình ảnh đơn lẻ
    """
    try:
        # Load và resize ảnh
        img = load_img(img_path, target_size=img_size)

        # Chuyển thành array và normalize
        img_array = img_to_array(img)
        img_array = img_array / 255.0  # Normalize về [0,1]
        img_array = np.expand_dims(img_array, axis=0)  # Thêm batch dimension

        # Dự đoán
        prediction = model.predict(img_array)
        probability = prediction[0][0]

        # Xác định class
        if probability > 0.5:
            predicted_class = "Cho phép chó vào"
            confidence = probability
        else:
            predicted_class = "Không được phép vào"
            confidence = 1 - probability

        return predicted_class, confidence, img
```

```
        except Exception as e:
            print(f"Lỗi khi xử lý ảnh {img_path}: {e}")
            return None, None, None


    def display_predictions(model, image_paths, titles=None):
        """
        Hiển thị kết quả dự đoán cho nhiều ảnh
        """
        n_images = len(image_paths)
        fig, axes = plt.subplots(1, n_images, figsize=(5*n_images, 5))

        if n_images == 1:
            axes = [axes]

        for i, img_path in enumerate(image_paths):
            predicted_class, confidence, img = predict_single_image(model, img_path)

            if predicted_class is not None:
                axes[i].imshow(img)
                axes[i].set_title(f'Predicted: {predicted_class}',
                                  fontsize=12, fontweight='bold')
                axes[i].axis('off')

                # Thêm title tùy chỉnh nếu có
                if titles and i < len(titles):
                    axes[i].set_xlabel(titles[i], fontsize=10)
            else:
                axes[i].text(0.5, 0.5, 'Error loading image'
```