Dataset Overview

The dataset used for training the CNN model consists of 25,000 images, with an equal distribution of dog and cat images (12,500 each). The images are sourced from Kaggle and are varied in terms of size and resolution. The dataset is used for binary classification, where the task is to distinguish between cats and dogs.

Statistics for the Images

I used a Python script that allowed me to extract information about each image individually and then compile overall statistics to assist with preprocessing later on.

```
import os
from PIL import Image

# Path to the directories containing the images
cat_dir = './cat'
dog_dir = './dog'

# Function to get image information
def get_image_info(directory):
    image_info = []
    for filename in os.listdir(directory):
        if filename.lower().endswith(('png', 'jpg', 'jpeg')):
            img_path = os.path.join(directory, filename)
            with Image.open(img_path) as img:
                width, height = img.size
                file_size = os.path.getsize(img_path)
                image_info.append({
                    'filename': filename,
                    'width': width,
                    'height': height,
                    'file_size': file_size,
                    'format': img.format
                })
    return image_info

# Get image information for both categories
cat_images_info = get_image_info(cat_dir)
dog_images_info = get_image_info(dog_dir)

# Print some details about the dataset
print(f"Number of cat images: {len(cat_images_info)}")
print(f"Number of dog images: {len(dog_images_info)}")
# Collect statistics about the images (e.g., average size, format)
def print_statistics(image_info, category):
    if not image_info:
        print(f"No images found for {category}.")
        return
```

```
    total_width = sum(info['width'] for info in image_info)
    total_height = sum(info['height'] for info in image_info)
    total_size = sum(info['file_size'] for info in image_info)

    avg_width = total_width / len(image_info)
    avg_height = total_height / len(image_info)
    avg_size = total_size / len(image_info)

    print(f"Statistics for {category}:")
    print(f"  Average width: {avg_width:.2f} px")
    print(f"  Average height: {avg_height:.2f} px")
    print(f"  Average file size: {avg_size / 1024:.2f} KB")

# Print statistics for cat and dog images /// remove the comment so
you can verify urself
# print_statistics(cat_images_info, "Cat")
# print_statistics(dog_images_info, "Dog")

Number of cat images: 12499
Number of dog images: 12499
```

Cat Images:

- **Average Width:** 410.84 px
- **Average Height:** 356.94 px
- **Average File Size:** 30.35 KB

Dog Images:

- **Average Width:** 398.06 px
- **Average Height:** 365.04 px
- **Average File Size:** 35.98 KB

These statistics indicate that the cat images are generally slightly wider than the dog images, while the dog images are slightly taller on average. The file sizes are also marginally larger for dog images, likely due to subtle differences in image content and compression.

Model break down

Libraries

Import necessary libraries such as:

- **os:** For directory and file path operations.
- **numpy:** For numerical computations.
- **seaborn:** For visualizations (e.g., confusion matrix).
- **matplotlib:** For plotting graphs.
- **tensorflow:** For building and training the deep learning model.
- **sklearn:** For metrics like classification report and ROC curve.

```
# Import Libraries
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.image import imread
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, MaxPooling2D, Dropout,
Flatten, BatchNormalization, Conv2D
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

Hyperparameters

Set key training parameters so can be chnaged easily.

```
# Data Directory Path
train_path = 'db/train/'
val_path = 'db/val/'
test_path = 'db/test/'

# Set Parameters
image_size = 128 # new image size for resizing
bat_size = 32 # Batch size for training and test then finally
validation
```

Data Augmentation and Preprocessing

Here the data will submit to changes to be ready for training and validation.

- **Rescaling:** Pixel values normalized to [0, 1].
- **Rotation:** Random rotations up to 15 degrees.
- **Horizontal flip**.
- **Zoom**, **shear**, and **shift** transformations.

```
# Data Augmentation for Training Set and Rescaling for Validation/Test
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.1,
```

```
    fill_mode='reflect',
    width_shift_range=0.1, # image shifted horizontally along the x-
axis.
    height_shift_range=0.1 # image shifted by a percentage of the
total height.
)
```

Data Generators

Data is loaded using the flow_from_directory method:

- **train_generator:** Loads training data with augmentation.
- **val_generator:** Loads validation data without augmentation.
- **test_generator:** Loads test data without augmentation and with shuffle=False to preserve order.

```
test_datagen = ImageDataGenerator(rescale=1./255)

# Load Data
train_generator = train_datagen.flow_from_directory(
    train_path,
    class_mode='binary',
    target_size=(image_size, image_size),
    batch_size=bat_size
)

val_generator = test_datagen.flow_from_directory(
    val_path,
    class_mode='binary',
    target_size=(image_size, image_size),
    batch_size=bat_size
)

test_generator = test_datagen.flow_from_directory(
    test_path,
    class_mode='binary',
    target_size=(image_size, image_size),
    batch_size=bat_size,
    shuffle=False
)
```

```
Found 17498 images belonging to 2 classes.
Found 3750 images belonging to 2 classes.
Found 3750 images belonging to 2 classes.
```

sh between two classes (e.g., cats and dogs).

Build CNN Model

To define a convolutional neural network (CNN) architecture for binary classification, i used these params:

Explanation of CNN Model Architecture

- **Conv2D**
- Extract spatial features from the input images by applying convolution filters.
- The number of filters increases progressively (32 → 64 → 128) to capture more complex features as the network deepens.
- **BatchNormalization**
- Normalizes the output of the previous layer to stabilize learning and speed up convergence.
- **MaxPooling2D**
- Reduces the spatial dimensions of feature maps to lower computational costs and retain the most important features.
- Helps make the model more efficient and robust to small image translations.
- **Dropout**
- Randomly deactivates a fraction of neurons during training to prevent overfitting.
- The dropout rate is 0.2 in convolutional blocks and 0.5 in the dense layers for stronger regularization.
- **Flatten**
- Converts the 2D feature maps from the convolutional layers into a 1D vector.
- Prepares the data for fully connected layers.
- **Dense Layers**
- Fully connected layers that use the flattened features to make decisions.
- The first dense layer (512 neurons) learns complex representations of the features.
- **Sigmoid Activation (Output Layer)**
- Outputs a single probability value between 0 and 1 for binary classification.

My previous model had more layers with **GlobalAveragePooling2D**, but its results were less efficient compared to using **MaxPooling2D**.

```
# CNN Model Architecture
model = Sequential()

# First Convolutional Block
model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(image_size, image_size, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Second Convolutional Block
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Third Convolutional Block
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Flatten and Fully Connected Layer
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Output Layer
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
C:\anaconda\Lib\site-packages\keras\src\layers\convolutional\
base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d (Conv2D)                  │ (None, 126, 126, 32)   │
896 │
```

```
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization             │ (None, 126, 126, 32)   │
128 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 63, 63, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout (Dropout)               │ (None, 63, 63, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_1 (Conv2D)               │ (None, 61, 61, 64)     │
18,496 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_1           │ (None, 61, 61, 64)     │
256 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 30, 30, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_1 (Dropout)             │ (None, 30, 30, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_2 (Conv2D)               │ (None, 28, 28, 128)    │
73,856 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_2           │ (None, 28, 28, 128)    │
512 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 14, 14, 128)    │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_2 (Dropout)             │ (None, 14, 14, 128)    │
```

```
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ flatten (Flatten)                │ (None, 25088)           │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ dense (Dense)                    │ (None, 512)             │
12,845,568 │
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ batch_normalization_3            │ (None, 512)             │
2,048 │
│ (BatchNormalization)             │                         │
│
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ dropout_3 (Dropout)              │ (None, 512)             │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ dense_1 (Dense)                  │ (None, 1)               │
513 │
└──────────────────────────────────┴─────────────────────────┴──────────
───────┘

 Total params: 12,942,273 (49.37 MB)

 Trainable params: 12,940,801 (49.37 MB)

 Non-trainable params: 1,472 (5.75 KB)
```

note: I used flatten at first since it simply converts a tensor into a one-dimensional vector, maintaining all its values. But according to what i h've been seeing in the internet, Global Average Pooling reduces the tensor's size by averaging its values, which can help to reduce overfitting by summarizing the feature maps, but reuslts were much worse so i had to use flatten again.

Compile the model

Used **adam optimizer** to have adaptive learning rate optimizer, effective for deep learning, also **binary crossentropy** appropriate for binary classification tasks.

```python
# Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Dynamic Learning Rate and Early Stopping

1. ReduceLROnPlateau

Reduces the learning rate when validation performance plateaus.

- **Parameters:**
  - `monitor='val_accuracy'`: Tracks validation accuracy.
  - `patience=2`: Waits for 2 epochs without improvement before reducing the learning rate.
  - `factor=0.5`: Halves the current learning rate.
  - `min_lr=0.00001`: Sets a minimum learning rate limit.
  - `verbose=1`: Outputs a message when the learning rate is redatio##n.

2. EarlySPurpose:** Stops training early to avoid overfitting and save time.

- **Parameters:**
  - `monitor='val_loss'`: Tracks validation loss.
  - `patience=3`: Allows 3 epochs without improvement before stopping.
  - `restore_best_weights=True`: Restores weights from the best epoch.
  - `verbose=0`: Suppresses outhe best weights.

```
# Callbacks for Dynamic Learning Rate and Early Stopping
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
patience=2, factor=0.5, min_lr=0.00001, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True, verbose=0)
```

Train the Model & save it

CNN using the training data while evaluating on the validation data.

```
# Train the Model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping, learning_rate_reduction]
)
```

```
# Save the Model
model.save("cat_dog_classifier.h5")
```

Evaluate the Model

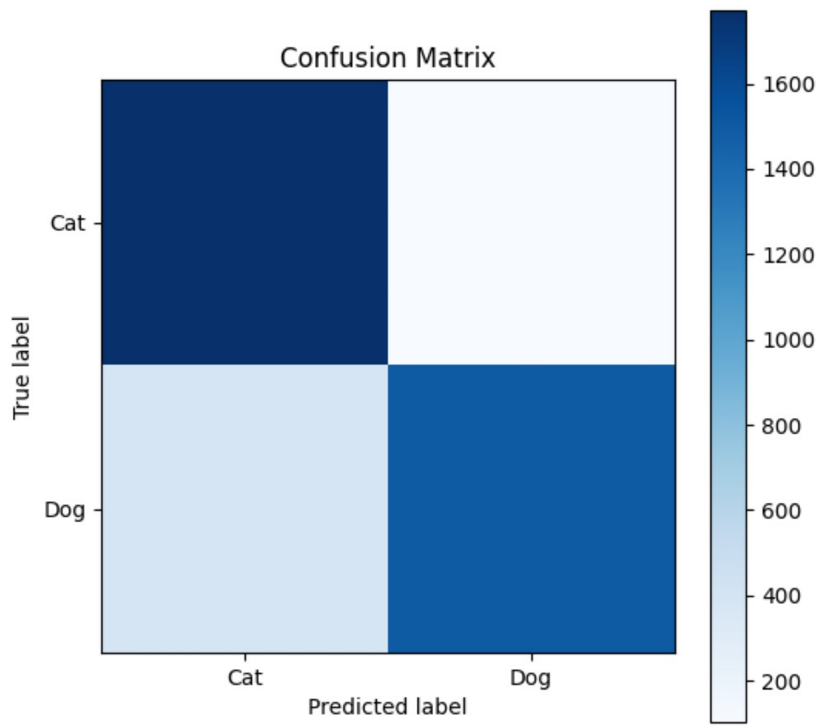
Evaluate the model's performance using metrics like confusion matrix and ROC-AUC, all results saved into file named final results using this code:

```
result = model.predict(test_generator, batch_size=bat_size, verbose=0)
y_pred = np.round(result).astype(int)  # 0 or 1
y_true = test_generator.labels

print(classification_report(y_true, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)

# Create a directory to save results
output_dir = 'final_results'
os.makedirs(output_dir, exist_ok=True)

# Save Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Cat',
'Dog'], yticklabels=['Cat', 'Dog'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.savefig(os.path.join(output_dir, 'confusion_matrix.png'))
plt.close()

# Save Classification Report
report = classification_report(y_true, y_pred, target_names=['Cat',
'Dog'], output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df.to_csv(os.path.join(output_dir,
'classification_report.csv'), index=True)

# Compute ROC Curve
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_true, result)
roc_auc = auc(fpr, tpr)

# Save ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--',
```

```
label='Random Guess')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.savefig(os.path.join(output_dir, 'roc_curve.png'))
plt.close()

# Save Training History
history_df = pd.DataFrame(history.history)
history_df.to_csv(os.path.join(output_dir, 'training_history.csv'),
index=False)

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
def display_txt_file_contents(file_path):
    try:
        with open(file_path, 'r') as file:
            contents = file.read()  # Read the entire file
            print(contents)  # Print the file contents
    except FileNotFoundError:
        print(f"The file at {file_path} was not found.")
    except IOError:
        print("An error occurred while reading the file.")

# Example usage
file_path = './final_results/classification_report.txt'  # Replace
with the actual path to your .txt file
display_txt_file_contents(file_path)
def display_images_from_folder(folder_path):
    images = [f for f in os.listdir(folder_path) if
f.endswith(('.png', '.jpg', '.jpeg'))]
    plt.figure(figsize=(35, 35))

    for i, image_file in enumerate(images):
        img_path = os.path.join(folder_path, image_file)
        img = mpimg.imread(img_path)

        plt.subplot(len(images), 1, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(image_file)

    plt.tight_layout()
    plt.show()
folder = './final_results'
display_images_from_folder(folder)
```

```
              precision    recall  f1-score   support

           0       0.82      0.95      0.88      1875
           1       0.94      0.80      0.86      1875

    accuracy                           0.87      3750
   macro avg       0.88      0.87      0.87      3750
weighted avg       0.88      0.87      0.87      3750
```

confusion_matrix.png



precision_recall_curve.png