College of Computer and Information Sciences
Computer Science Department

Machine Learning CSC 462

Prepared by:

Danah Alomran	443201027
Asma Albahkly	443200597
Shahad Almutairi	442201347

First Semester 1446

I. Selection of image classification task and suitable dataset

In this project we are choosing binary image classification: distinguishing between images of cats and dogs. Our goal is to develop a model that can correctly identify if a given image is a cat or a dog, which can be beneficial in applications like automated pet identification systems or organizing image libraries.

The classification task has two clear and mutually exclusive categories:

1. **Cat**: Represented by images labeled as "cat."
2. **Dog**: Represented by images labeled as "dog."

Each image in the dataset belongs to one of these two categories, and our model's objective is to correctly classify an image into one of them. The simplicity of having just two classes makes this a well-suited problem for binary classification using Convolutional Neural Networks (CNNs).

For this task, the chosen dataset is the **Cats vs. Dogs** dataset from Kaggle[1] by Anthony Therrien, designed for binary image classification tasks. It consists of 1,000 images, with 500 each for cats and dogs, organized into respective folders. The images showcase a wide variety of breeds, poses, and backgrounds, including indoor and outdoor settings, making it a challenging dataset. Each image is labeled as 0 for cats and 1 for dogs.

II. Review of literature

Summarized by: Asma Albahkly

Adriyanto et al. (2022) [2] investigate the use of Convolutional Neural Networks (CNNs) for binary image classification, specifically in distinguishing cats from dogs, with a focus on potential applications for assisting the visually impaired. Their CNN architecture includes three convolutional layers with 16, 32, and 64 filters, respectively, each followed by ReLU activation functions. Max-pooling is applied to downsample the image size after each convolution. The model is designed to process 150x150 pixel RGB images and employs dropout regularization (0.5) to reduce overfitting. After feature extraction, the model flattens the data and connects it to a fully connected layer with 512 nodes. The final output layer uses a sigmoid activation function for binary classification (dog or cat), optimized using binary cross-entropy loss and the Adam optimizer with a learning rate of 0.001 [2].

The model was trained on a dataset of 1,000 images for both training and validation, with an additional 78 images (51 dog and 27 cat) used for testing. After training for 20 epochs, the model achieved 64% accuracy. The performance metrics, including 45% precision, recall, and F1-score, suggest that the model's ability to generalize to new data is limited, pointing to areas for improvement in accuracy and robustness [2].

The authors propose that further optimization, such as incorporating data augmentation techniques (e.g., rotations, flips), fine-tuning hyperparameters, and exploring deeper network architectures like ResNet or DenseNet, could improve results. Despite the modest performance, the study highlights CNNs' potential for real-world applications, particularly in assistive technology for the visually impaired [2].

Summarized by: Danah Alomran

In [3] the Kaggle “Cat vs Dog” dataset, containing 25,000 images equally split between cats and dogs, was used for training CNN models. During preprocessing, images were resized to 32x32 and 64x64 pixels to match CNN input requirements and normalized to a [0, 1] range to improve model performance. The CNN architecture included convolutional layers with ReLU activation for feature extraction, max pooling layers to reduce parameters while preserving key features, and fully connected layers for classification. Some models used Dropout to reduce overfitting and enhance generalization.

Eight models were trained for 30 epochs, varying in the number of layers and filters (32, 64, 128) with a 3x3 kernel size. The highest accuracy, 99.26%, was achieved by a 3-layer CNN with 32x32 input images and Dropout enabled. Models with 64x64 input sizes performed worse, likely due to insufficient data to match the model’s complexity, while simpler 2-layer architectures also underperformed. Dropout significantly contributed to improved performance by preventing overfitting.

Summarized by: Shahad Almutairi

This study [4] used a dataset of 21 dog and cat breeds, comprising 20,574 images for training, 2,572 for validation, and 2,590 for testing. The dataset included breeds like Akita and Siamese Cat. Images were resized to 224x224 pixels to fit the VGG model, with the top layer removed and fully connected layers redefined for the new categories. Stochastic Gradient Descent (SGD) with a learning rate of 0.0001 and binary cross-entropy loss was used, training for 50 epochs with a batch size of 16.

The architecture was based on VGG16 and VGG19, both with stacked convolutional layers and max-pooling for dimensionality reduction. VGG16 had 13 convolutional layers, while VGG19 had 16. The depth of the network increased with filters from 64 to 512. VGG16 achieved a training accuracy of 98.47%, validation accuracy of 98.56%, and testing accuracy of 83.68%, while VGG19 achieved slightly better testing accuracy (84.07%) [4].

The results confirm that fine-tuned VGG16 and VGG19 models perform well for classifying dog and cat breeds, with VGG19 slightly outperforming VGG16 in testing accuracy, highlighting the benefit of deeper networks for detailed feature extraction [4].

III. Design of the CNNs

Our CNN model for classifying images of cats and dogs starts with an input layer, where images are resized to either 128x128 or 150x150 pixels, and pixel values are normalized to the [0, 1] range. The architecture includes three convolutional layers: the first has 32 filters with a 3x3 kernel size, a stride of 1, and same padding, followed by ReLU activation, max-pooling (commonly set as 2x2 based on the literature), and an optional dropout layer. The second convolutional layer has 64 filters with similar settings, using ReLU activation, max-pooling, and an optional dropout. The third convolutional layer uses 128 filters, also with a 3x3 kernel, ReLU activation, max-pooling (e.g., 2x2), and an optional dropout. The output from these layers is flattened into a 1D vector for fully connected layers. The first fully connected layer consists of 512 units with ReLU activation, and we will consider incorporating dropout and batch normalization based on performance needs. The output layer features a single unit with sigmoid activation for binary classification. The model employs the Adam optimizer, and we will experiment with learning rates such as 0.01, 0.001, and 0.0001, alongside binary cross-entropy loss. Training will be conducted with a batch size of 32 over 20-30 epochs, using early stopping to prevent overfitting, and we will use cross-validation to evaluate model performance. We will also track metrics such as accuracy, precision, recall, and F1-score.

I. Model Strengths

Our CNN model achieved excellent performance by effectively extracting complex features through its layered architecture with progressively larger filter sizes. This design allowed the model to capture both simple and detailed patterns in the images, significantly boosting classification accuracy. To enhance generalization and prevent overfitting, we employed regularization techniques like dropout and batch normalization. Dropout helped reduce reliance on specific neurons, encouraging better generalization, while batch normalization improved training stability and accelerated convergence.

Additionally, early stopping was used to optimize training efficiency, halting the process when performance on the validation set began to decline, preventing overfitting. To ensure reliable evaluation, we applied 5-fold cross-validation, which allowed the model to be tested on different data subsets, providing a more accurate measure of its generalization ability.

The Adam optimizer further contributed to efficient convergence by adjusting the learning rate dynamically, leading to faster training without sacrificing accuracy. Together, these strategies ensured that the model was both highly accurate and computationally efficient, offering strong performance while avoiding common pitfalls like overfitting.

II. Methodology

Based on the literatures, we selected Convolutional Neural Networks (CNNs) for the task of classifying cat and dog images due to their efficiency in feature extraction. The data was preprocessed by resizing images to 128×128 pixels and normalizing pixel values to the range [0,1]. The CNN architecture consists of three convolutional layers with filters (32, 64, 128), ReLU activation, max-pooling for dimensionality reduction, and dropout rate for regularization. This is followed by a dense layer with 512 units and batch normalization, and a final output layer with a single sigmoid-activated unit for classification.
The model was trained using 5-fold cross-validation to ensure robust evaluation, employing the Adam optimizer with learning rates (0.01, 0.001, 0.0001) and binary cross-entropy loss. Metrics such as accuracy, precision, recall, and F1-score were used for performance evaluation, while early stopping was applied to prevent overfitting.

III. Results

Our experiments showed that the model performed best with a learning rate of 0.001. This learning rate provided a good balance between fast convergence and stable training, preventing issues such as overshooting or slow learning. We also tested different filter sizes in the convolutional layers, including 16, 32, and 64 filters. However, these changes resulted in a decrease in performance. It seems that smaller filter sizes were unable to capture enough important features, which affected the model's ability to distinguish patterns in the images.

Additionally, we tried reducing the number of units in the fully connected layer to fewer than 512. This also caused a drop in performance, as the smaller number of units limited the model's ability to learn complex patterns. A larger fully connected layer allowed the model to capture more detailed relationships in the data, improving its overall accuracy.

After optimizing these parameters, our model achieved an accuracy of 97.48%, outperforming the baseline CNN model we built, which achieved 94.30%. These results show that our design and parameter choices were effective in improving the model's performance and ability to generalize.

Learning rate	Accuracy	Precision	Recall	F1-Score
0.01	0.926	0.9244	0.936	0.930
0.001	0.974	0.9810	0.9674	0.9741
0.0001	0.8428	0.9674	0.7084	0.8178

IV. Limitations

A potential limitation of our CNN model lies in its complexity, which may not be well-suited to the relatively small size of our dataset, consisting of only 1,000 images. The model's intricate architecture, with multiple convolutional layers and a large number of units in the fully connected layer, increases the risk of overfitting, as it may learn noise rather than generalizable patterns due to the limited data.

While the model achieved strong accuracy on both the training and test sets, there were instances where the validation accuracy was lower than the training accuracy during certain epochs, suggesting overfitting. However, the model still showed consistently good validation accuracy across all epochs, indicating reasonable generalization despite its complexity.

In conclusion, while the model's complexity posed a risk of overfitting due to the small dataset size, the regularization techniques helped mitigate this issue, allowing for good generalization.

V. Future Improvements

Future improvements for the CNN model could involve data augmentation to expand the dataset diversity, model simplification to reduce the risk of overfitting, and incorporating additional regularization techniques such as L2 regularization or increased dropout. Exploring advanced architectures like ResNet or EfficientNet, along with leveraging transfer learning from pre-trained models, could further enhance performance. These modifications would help optimize the model's accuracy and generalization, particularly for smaller datasets, while reducing the likelihood of overfitting.

Screenshots of the code:

```python
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import KFold

# Dataset path
dataset_path = '/Users/danahalomran/Documents/animals'


data_gen = ImageDataGenerator(rescale=1./255)

data_generator = data_gen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    shuffle=True  # Shuffle the data
)


X, y = [], []
for i in range(len(data_generator)):
    batch_data, batch_labels = data_generator[i]
    X.extend(batch_data)
    y.extend(batch_labels)
X = np.array(X)
y = np.array(y)
```

```python
# CNN model creation function
def create_model(learning_rate):
    model = keras.Sequential([
        keras.layers.InputLayer(input_shape=(128, 128, 3)),

        keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Dropout(0.2),

        keras.layers.Flatten(),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(1, activation='sigmoid')


    ])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss='binary_crossentropy',
        metrics=['accuracy', 'Precision', 'Recall']
    )
    return model
```

```python
# K-Fold Cross-Validation
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

fold = 1
for train_index, val_index in kf.split(X):
    print(f"Training on Fold {fold}...")

    # Split the data into training and validation sets
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Create the model
    model = create_model(learning_rate=0.01)

    # Early stopping
    early_stopping = EarlyStopping(patience=5, restore_best_weights=True)

    # Train the model
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=30,
        batch_size=32,
        callbacks=[early_stopping]
    )

    # Evaluate the model
    val_loss, val_accuracy, val_precision, val_recall = model.evaluate(X_val, y_val)
    print(f"Fold {fold} - Validation Loss: {val_loss}, Accuracy: {val_accuracy}, Precision: {val_precision}, Recall: {val_recall}")

    fold += 1
```

VI. References

[1] https://www.kaggle.com/datasets/anthonytherrien/dog-vs-cat

[2] T. Adriyanto, R. A. Ramadhani, R. Helilintar, and A. Ristyawan, "Classification of dog and cat images using the CNN method," *Ilk. J. Ilm*, vol. 14, no. 3, pp. 203–208, 2022.

[3] A. Sarraf, "Binary Image Classification Through an Optimal Topology for Convolutional Neural Networks," vol. 68, no. 1, 2020.

[4] Mahardi, I.-H. Wang, K.-C. Lee, and S.-L. Chang, "Images Classification of Dogs and Cats using Fine-Tuned VGG Models," in *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, Yunlin, Taiwan: IEEE, Oct. 2020, pp. 230–233. doi: 10.1109/ECICE50847.2020.9301918.