Binary Classification

Eric Zhuang & Kan Zhang

Contents

Personal Statement

- Binary classification categorizes data into two classes using learned patterns.

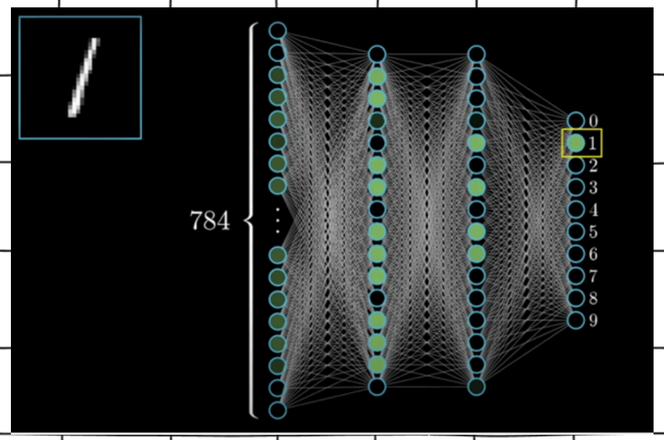
- Challenges include accuracy, imbalanced datasets, and biases.

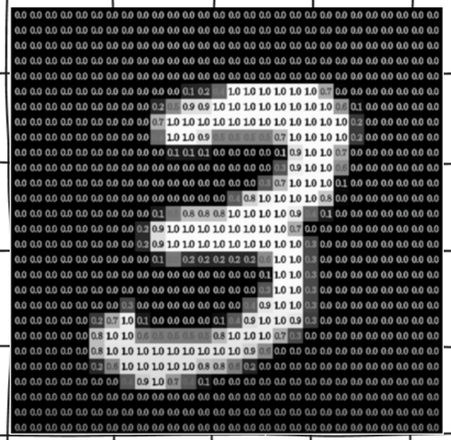
- Our study focuses on optimizing a cat vs. dog classification model.

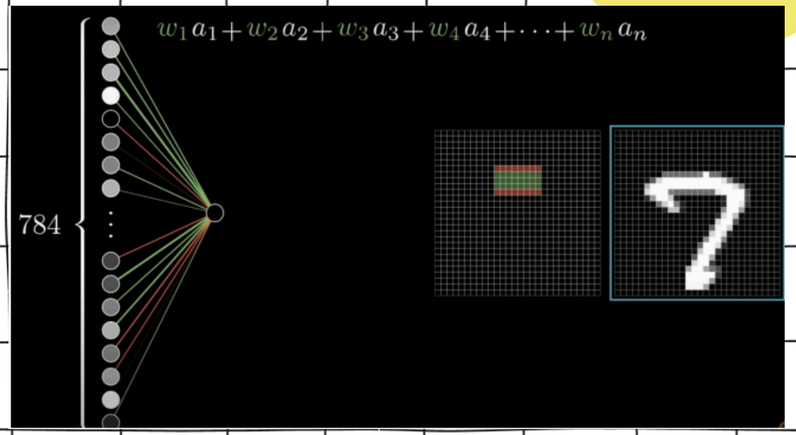
- We analyze parameter adjustments' impact on training and performance.

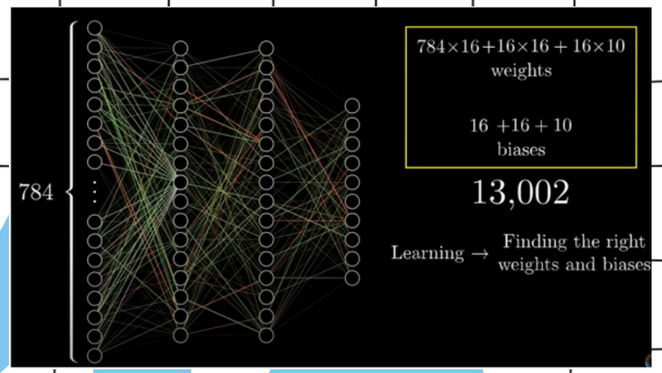
- Key insights inform crucial factors for model reliability.

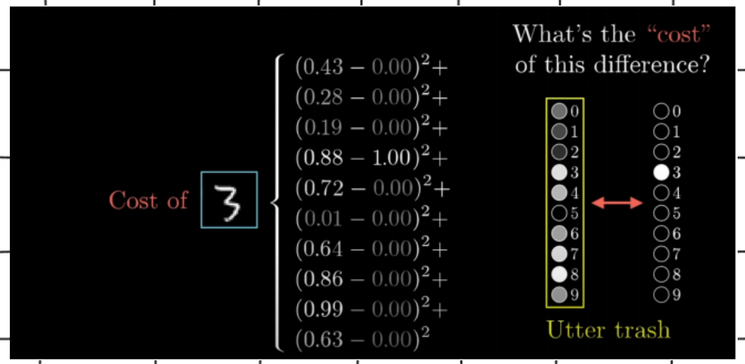
Neural Network

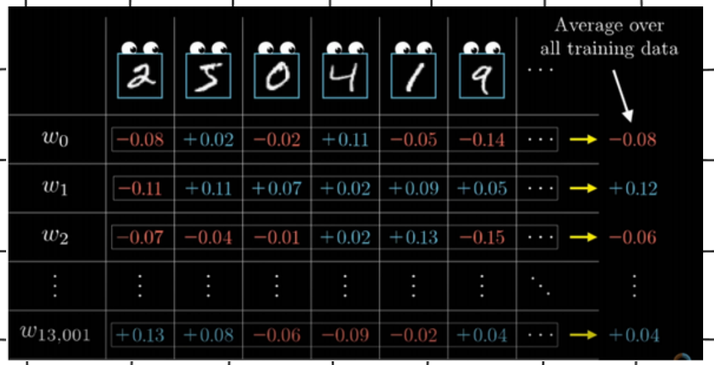












Backpropagation

$$a^{(L)} = \sigma\left(w^{(L)}a^{(L-1)} + b^{(L)}\right)$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)}\sigma'\left(z^{(L)}\right)2\left(a^{(L)} - y\right)$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_0}{\partial a^{(L)}} = 1\sigma'\left(z^{(L)}\right)2\left(a^{(L)} - y\right)$$

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_0}{\partial a^{(L)}} = w^{(L)}\sigma'\left(z^{(L)}\right)2\left(a^{(L)} - y\right)$$

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_l-1}\frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C_0}{\partial a_j^{(L)}}$$

CNN

Convolutional Neural Network

01

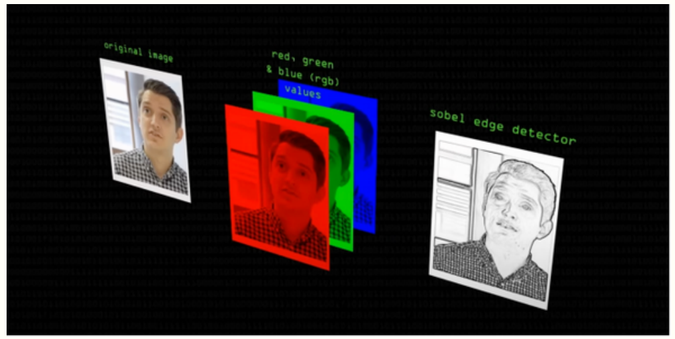
The Sobel operator efficiently detects horizontal edges by assigning intensity values from 0 to 255. In convolutional neural networks (CNNs), multiple kernels act as filters, extracting features like edges and corners across RGB channels. These kernels generate initial feature maps, which are refined in subsequent layers to capture increasingly complex patterns.

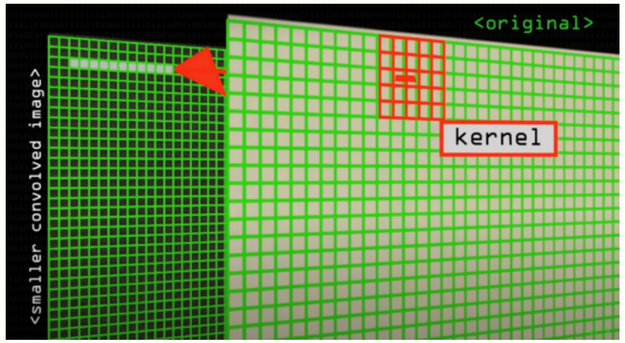
02

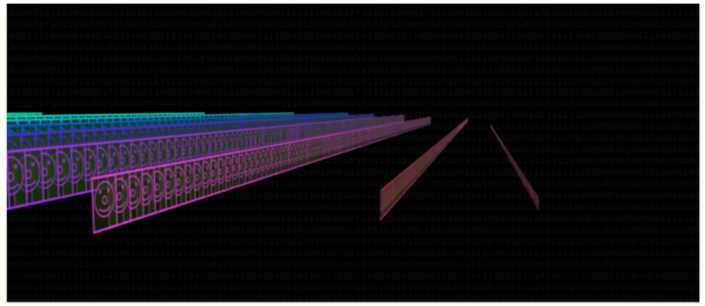
Kernels, represented as matrices (e.g., 3x3), slide over each pixel, producing a single output by multiplying and summing kernel values with RGB intensities. This convolution process reduces feature map dimensions while enhancing feature extraction efficiency across CNN layers.

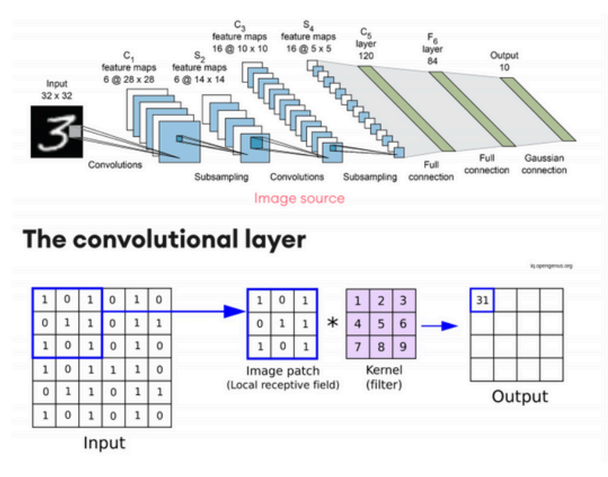
03

CNNs progress from detecting basic features like edges and corners in early layers to combining these features into more intricate patterns in later layers. This hierarchical approach allows networks to learn nuanced representations essential for tasks such as image recognition and object detection.









Code Analysis

```
# importing libraries
import numpy as np
import random
import matplotlib.pyplot as plt
# %matplotlib inline  # Uncomment this if you're running in a Jupyter notebook
from sklearn.metrics import confusion_matrix
import seaborn as sns
sns.set(style='darkgrid', font_scale=1.4)
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

```
# Tensorflow


from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
# Paths
train_dir = '/Users/a123/Desktop/ERIC/train'
test_dir = '/Users/a123/Desktop/ERIC/test'

# Hyperparameters
CFG = {'seed': 77, 'batch_size': 16, 'img_size': (299, 299), 'epochs': 5, 'patience': 5}
```

```python
# Data Augmentation
train_data_generator = ImageDataGenerator(
    validation_split=0.15,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    preprocessing_function=preprocess_input,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```python
train_generator = create_data_generator(train_dir, train_data_generator, 'training')
validation_generator = create_data_generator(train_dir, val_data_generator, 'validation')
test_generator = create_data_generator(test_dir, test_data_generator, None)

# Number of samples and classes
nb_train_samples = train_generator.samples
nb_validation_samples = validation_generator.samples
nb_test_samples = test_generator.samples
classes = list(train_generator.class_indices.keys())
num_classes = len(classes)
print('Classes:', classes)
```

```python
# Visualize some examples
plt.figure(figsize=(15, 15))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    ax.grid(False)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    batch = next(train_generator)
    imgs = (batch[0] + 1) * 127.5
    label = np.argmax(batch[1][0])  # Adjusted to handle one-hot encoded labels
    image = imgs[0].astype('uint8')
    plt.imshow(image)
    plt.title(classes[label])  # Use the class names instead of hardcoding
plt.show()
```

```
# Pre-trained deep convolutional neural network
base_model = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(CFG['img_size'][0], CFG['img_size'][1], 3))
```

```
# Add new layers
x = base_model.output
x = Flatten()(x)
x = Dense(100, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax', kernel_initializer='random_uniform')(x)
```

```
# Define optimizer
optimizer = Adam()
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Save the best model
save_checkpoint = ModelCheckpoint(filepath='model.h5.keras', monitor='val_loss', save_best_only=True, verbose=1)
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=CFG['patience'], verbose=True)
```

```
# Train model
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // CFG['batch_size'],
    epochs=CFG['epochs'],
    callbacks=[save_checkpoint, early_stopping],
    validation_data=validation_generator,
    verbose=True,
    validation_steps=nb_validation_samples // CFG['batch_size']
```

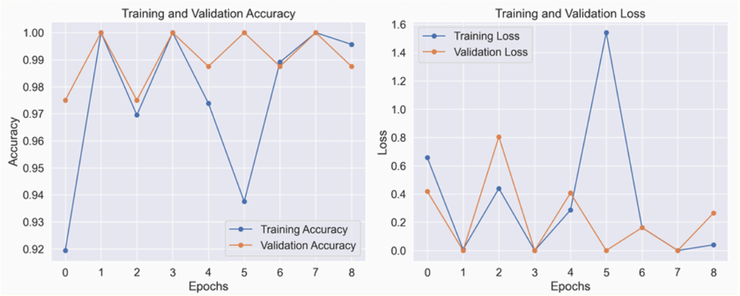
```python
# Accuracy and Loss Curves
def plot_accuracy_and_loss(history):
    plt.figure(figsize=(15, 6))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss', marker='o')
    plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

plot_accuracy_and_loss(history)
```

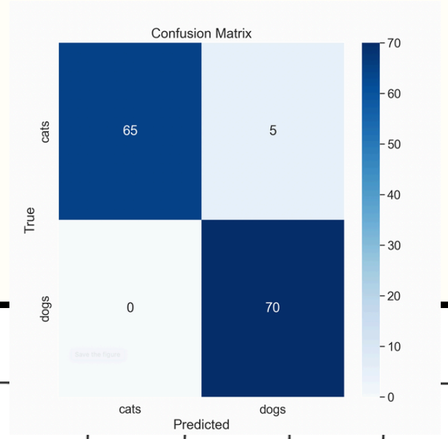
```python
# Confusion Matrix
def plot_confusion_matrix(model, test_generator):
    y_true = test_generator.classes
    y_pred = model.predict(test_generator)
    y_pred = np.argmax(y_pred, axis=1)

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

plot_confusion_matrix(model, test_generator)
```

Changing Parameters

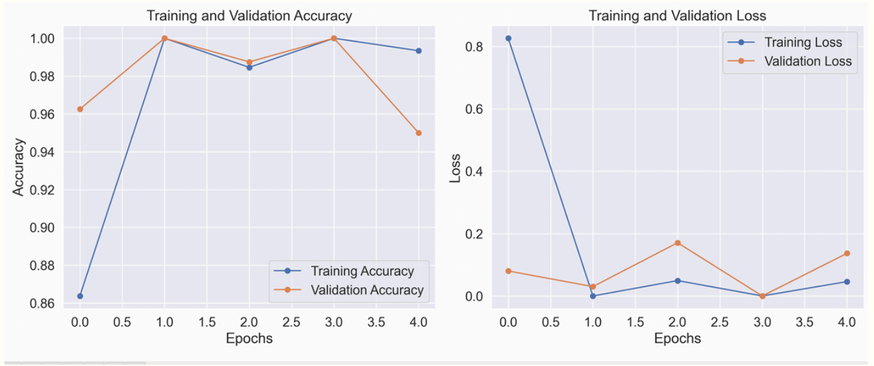
epoch, batch size, # neurons

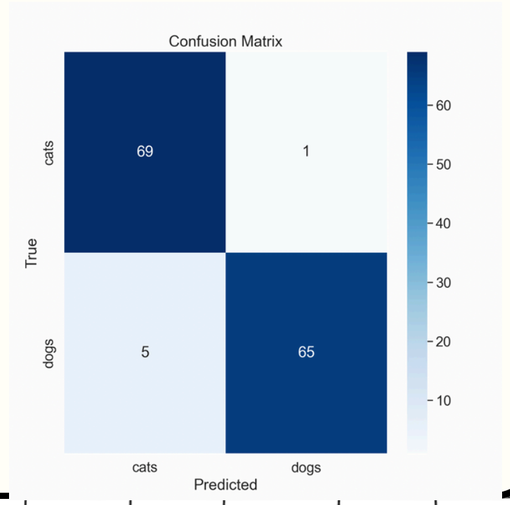
epochs = 10



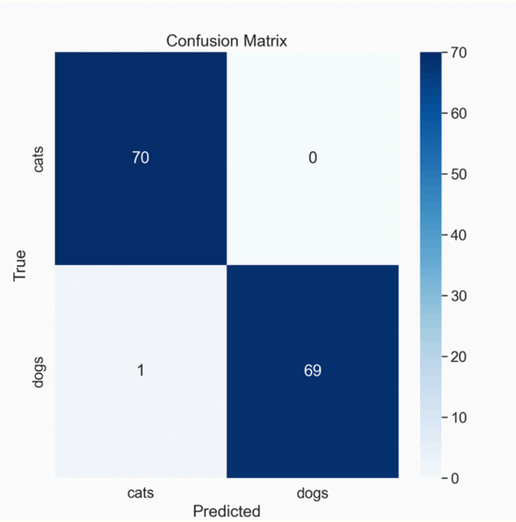


batch size = 20





x = Dense(50, activation='relu')(x)



Conclusions

Conclusion 01

Optimal epoch = 5

epoch = 3: under-fitting

epoch = 10: over-fitting

Conclusion 02

Optimal batch size ≈ 16

batch size = 20: unequal # of cat & dog images

Conclusion 03

Optimal # of neurons ≈ 100

# of neurons = 50: not capturing enough

# of neurons = 150: useless characteristics

Thank you

www.kansucks.com