UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Trabalho Prático - Gato vs Não-Gato

Matheus Henrique Schaly (18200436)
Pedro Alonso Condessa (19200360)

Pipeline

Os passos realizados no Google Colab foram:

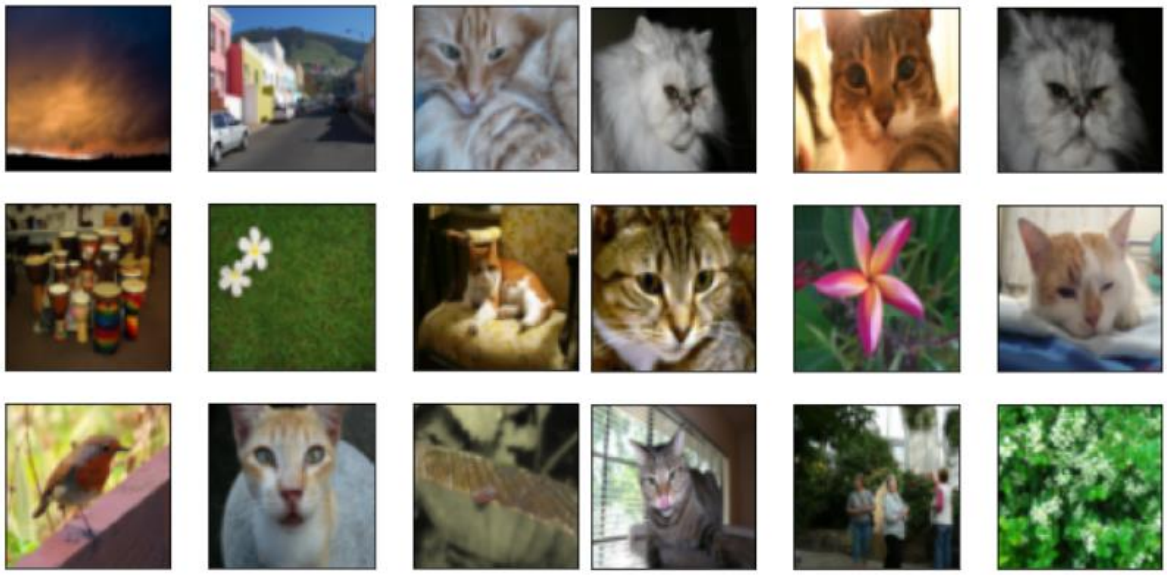
1. Importação das bibliotecas.
2. Configuração das seeds.
3. Carregamento dos dados.
4. Visualização das imagens.
5. Normalização dos dados.
6. Treinamento dos modelos:
    a. Logistic Regression
    b. Perceptron
    c. Shallow Feedforward Neural Network
    d. Convolutional Neural Network

Todos os modelos seguem o seguinte pipeline:

1. Possível configuração dos hiperparâmetros.
2. Treinamento.
3. Previsão.
4. Resultados.

Visualização de algumas imagens

Abaixo temos algumas imagens do conjunto de treino e algumas do conjunto do conjunto de teste.



Logistic Regression

Os hiperparâmetros utilizados são os apresentados abaixo. É criado uma combinação entre cada elemento dessas listas. Neste caso temos 2*5*2 = 20 modelos treinados.

```
'penalty': ['l1', 'l2'],
'C': [0.001, 0.01, 1, 10, 100],
'solver': ["newton-cg", "lbfgs"]
```

Os parâmetros utilizados no melhor modelo encontrado, seguido pela acurácia no conjunto de treino foi:

```
{'C': 0.001, 'penalty': 'l2', 'solver': 'newton-cg'}
0.6221835075493612
```

Os resultados obtidos no conjunto de teste foram:

```
Confusion matrix:
 [[17  0]
 [31  2]]
Accuracy: 0.38
Precision: 1.0
Recall: 0.0606
F1-Score: 0.1143
              precision    recall  f1-score   support

           0       0.35      1.00      0.52        17
           1       1.00      0.06      0.11        33

    accuracy                           0.38        50
   macro avg       0.68      0.53      0.32        50
weighted avg       0.78      0.38      0.25        50
```

Perceptron

Os hiperparâmetros utilizados foram:

```
'eta0': [0.0001, 0.001, 0.01, 0.1, 1.0],
'alpha': [0.0001, 0.001, 0.01, 0.1, 1.0]
```

Os parâmetros utilizados no melhor modelo encontrado, seguido pela acurácia no conjunto de treino foi:

```
{'alpha': 0.0001, 'eta0': 0.001}
0.5786295005807202
```

Os resultados obtidos no conjunto de teste foram:

```
Confusion matrix:
 [[11  6]
 [11 22]]
Accuracy: 0.66
Precision: 0.7857
Recall: 0.6667
F1-Score: 0.7213
              precision    recall  f1-score   support

           0       0.50      0.65      0.56        17
           1       0.79      0.67      0.72        33

    accuracy                           0.66        50
   macro avg       0.64      0.66      0.64        50
weighted avg       0.69      0.66      0.67        50
```

Shallow Neural Network

A arquitetura utilizada foi:

```
[35] # Create callbacks
     callbacks = [EarlyStopping(monitor='val_loss', patience=100)]

     # Create architecture
     model = Sequential()
     model.add(Dense(256, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))
     adam = adam_v2.Adam(learning_rate=0.001)
     model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])

     # Fit network architecture
     history = model.fit(X_train, y_train, epochs=1000, batch_size=50, validation_split= 0.2,
                         verbose=2, callbacks=[callbacks])

     # Print model summary
     print(model.summary())
```

Os resultados obtidos no conjunto de teste foram:

```
Confusion matrix:
 [[11  6]
 [ 5 28]]
Accuracy: 0.78
Precision: 0.8235
Recall: 0.8485
F1-Score: 0.8358
              precision    recall  f1-score   support

           0       0.69      0.65      0.67        17
           1       0.82      0.85      0.84        33

    accuracy                           0.78        50
   macro avg       0.76      0.75      0.75        50
weighted avg       0.78      0.78      0.78        50
```

Convolutional Neural Network

A arquitetura utilizada foi:

```
[41] # Create callbacks
     callbacks = [EarlyStopping(monitor='val_loss', patience=100)]

     # Create architecture
     model = Sequential()
     model.add(Conv2D(filters=16, kernel_size=3, activation='relu', input_shape=(64, 64, 3)))
     model.add(MaxPool2D(pool_size=2, strides=1))
     model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
     model.add(MaxPool2D(pool_size=2, strides=1))
     model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
     model.add(MaxPool2D(pool_size=2, strides=1))
     model.add(Conv2D(filters=128, kernel_size=3, activation='relu'))
     model.add(MaxPool2D(pool_size=2, strides=1))
     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dense(1, activation='sigmoid')) # Binary
     adam = adam_v2.Adam(learning_rate=0.001)
     model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy']) # Binary

     # Fit network architecture
     history = model.fit(X_train, y_train, epochs=50, batch_size=50, validation_split=0.2,
                         verbose=2, callbacks=[callbacks])

     # Print model summary
     print(model.summary())
```

Os resultados obtidos no conjunto de teste foram:

```
Confusion matrix:
 [[15  2]
 [ 8 25]]
Accuracy: 0.8
Precision: 0.9259
Recall: 0.7576
F1-Score: 0.8333
              precision    recall  f1-score   support

           0       0.65      0.88      0.75        17
           1       0.93      0.76      0.83        33

    accuracy                           0.80        50
   macro avg       0.79      0.82      0.79        50
weighted avg       0.83      0.80      0.81        50
```

Conclusões

Vamos realizar as análises levando em consideração o F1-Score, visto que temos um certo desbalanceamento entre classes no nosso conjunto de treino (137 não gatos e 72 gatos).

O F1-Score dos modelos foram:

- Logistic Regression (LR): 0.1143
- Perceptron: 0.7213

- Shallow Neural Network (SNN): 0.8358
- Convolutional Neural Network (CNN): 0.833

Com isso temos que o modelo de SNN obtive o melhor resultado, não por uma margem tão larga. Contudo, isso não deveria ser o caso, dado que uma CNN certamente seria um melhor modelo para a tarefa em questão se comparado com os demais modelos.