PokemonNN

May 6, 2024

1 Data Import

[1]:
```python
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib
import numpy as np
```

```
2024-05-04 23:01:06.424034: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-05-04 23:01:06.425157: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-04 23:01:06.430255: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-04 23:01:06.494415: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-04 23:01:07.570969: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:
```python
pokemon = pd.read_csv('Pokemon.csv')
pokemon.head(5)
```

[2]:
```
   number                 name type1   type2  total  hp  attack  defense  \
0       1            Bulbasaur Grass  Poison    318  45      49       49
1       2              Ivysaur Grass  Poison    405  60      62       63
2       3             Venusaur Grass  Poison    525  80      82       83
3       3        Mega Venusaur Grass  Poison    625  80     100      123
4       3  Gigantamax Venusaur Grass  Poison    525  80      82       83
```

```
   sp_attack  sp_defense  speed  generation  legendary
0         65          65     45           1      False
1         80          80     60           1      False
2        100         100     80           1      False
3        122         120     80           1      False
4        100         100     80           1      False
```

[49]:
```python
legendary_pokemon = pokemon[pokemon['legendary'] == True]
legendary_pokemon.head(5)
```

[49]:
```
     number              name     type1   type2  total  hp  attack  defense  \
192     144          Articuno       Ice  Flying    580  90      85      100
193     144  Galarian Articuno   Psychic  Flying    580  90      85       85
194     145    Galarian Zapdos  Fighting  Flying    580  90     125       90
195     146           Moltres      Fire  Flying    580  90     100       90
196     146   Galarian Moltres      Dark  Flying    580  90      85       90

     sp_attack  sp_defense  speed  generation  legendary
192         95         125     85           1       True
193        125         100     95           8       True
194         85          90    100           8       True
195        125          85     90           1       True
196        100         125     90           8       True
```

[50]:
```python
pokemon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1068 entries, 0 to 1067
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   number      1068 non-null   int64
 1   name        1068 non-null   object
 2   type1       1068 non-null   object
 3   type2       572 non-null    object
 4   total       1068 non-null   int64
 5   hp          1068 non-null   int64
 6   attack      1068 non-null   int64
 7   defense     1068 non-null   int64
 8   sp_attack   1068 non-null   int64
 9   sp_defense  1068 non-null   int64
 10  speed       1068 non-null   int64
 11  generation  1068 non-null   int64
 12  legendary   1068 non-null   bool
dtypes: bool(1), int64(9), object(3)
memory usage: 101.3+ KB
```

2 Data Preparation

[2]:
```
df = pd.read_csv('Pokemon.csv')

# Step 1: Select relevant columns
X = df[['hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed']]
y = df['legendary']

# Step 2: No missing or inconsistent data assumed in this example

# Step 3: No categorical variables need to be converted in this example

# Step 4: Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Optional: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)
y_train = y_train.astype(int)
```

[7]:
```
X.head()
```

[7]:

	hp	attack	defense	sp_attack	sp_defense	speed
0	45	49	49	65	65	45
1	60	62	63	80	80	60
2	80	82	83	100	100	80
3	80	100	123	122	120	80
4	80	82	83	100	100	80

[53]:
```
y.head()
```

[53]:
```
0    False
1    False
2    False
3    False
4    False
Name: legendary, dtype: bool
```

3 Defining the Neural Network

[3]:
```
model = keras.Sequential([
    layers.Dense(49, activation='relu', input_shape=[6]),
    layers.Dense(49, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

```python
model.compile(
    optimizer="adam",
    loss='binary_crossentropy',
    metrics=['binary_accuracy']
)

model.summary()
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87:
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 #
dense (Dense)	(None, 49)	343
dense_1 (Dense)	(None, 49)	2,450
dense_2 (Dense)	(None, 1)	50

Total params: 2,843 (11.11 KB)

Trainable params: 2,843 (11.11 KB)

Non-trainable params: 0 (0.00 B)

[4]:
```python
history = model.fit(
    X_train, y_train,
    epochs=100,     # Number of epochs (iterations over the entire dataset)
    batch_size=32, # Number of samples per gradient update
    validation_data=(X_test, y_test) # Optional: Data used for validation␣
 ↪during training
)
```

```
Epoch 1/100
27/27              1s 10ms/step -
binary_accuracy: 0.8121 - loss: 0.5656 - val_binary_accuracy: 0.8837 - val_loss:
```

```
0.3665
Epoch 2/100
27/27            0s 4ms/step -
binary_accuracy: 0.9034 - loss: 0.3183 - val_binary_accuracy: 0.8837 - val_loss:
0.2993
Epoch 3/100
27/27            0s 3ms/step -
binary_accuracy: 0.9041 - loss: 0.2730 - val_binary_accuracy: 0.9023 - val_loss:
0.2679
Epoch 4/100
27/27            0s 4ms/step -
binary_accuracy: 0.9291 - loss: 0.2078 - val_binary_accuracy: 0.8977 - val_loss:
0.2503
Epoch 5/100
27/27            0s 3ms/step -
binary_accuracy: 0.9426 - loss: 0.1822 - val_binary_accuracy: 0.8977 - val_loss:
0.2354
Epoch 6/100
27/27            0s 4ms/step -
binary_accuracy: 0.9522 - loss: 0.1791 - val_binary_accuracy: 0.9116 - val_loss:
0.2234
Epoch 7/100
27/27            0s 3ms/step -
binary_accuracy: 0.9443 - loss: 0.1605 - val_binary_accuracy: 0.9209 - val_loss:
0.2147
Epoch 8/100
27/27            0s 3ms/step -
binary_accuracy: 0.9439 - loss: 0.1586 - val_binary_accuracy: 0.9256 - val_loss:
0.2093
Epoch 9/100
27/27            0s 4ms/step -
binary_accuracy: 0.9343 - loss: 0.1752 - val_binary_accuracy: 0.9349 - val_loss:
0.2002
Epoch 10/100
27/27            0s 3ms/step -
binary_accuracy: 0.9413 - loss: 0.1500 - val_binary_accuracy: 0.9349 - val_loss:
0.1950
Epoch 11/100
27/27            0s 3ms/step -
binary_accuracy: 0.9246 - loss: 0.1699 - val_binary_accuracy: 0.9442 - val_loss:
0.1890
Epoch 12/100
27/27            0s 3ms/step -
binary_accuracy: 0.9389 - loss: 0.1335 - val_binary_accuracy: 0.9442 - val_loss:
0.1845
Epoch 13/100
27/27            0s 3ms/step -
binary_accuracy: 0.9479 - loss: 0.1556 - val_binary_accuracy: 0.9442 - val_loss:
```

```
0.1802
Epoch 14/100
27/27             0s 3ms/step -
binary_accuracy: 0.9281 - loss: 0.1687 - val_binary_accuracy: 0.9442 - val_loss:
0.1756
Epoch 15/100
27/27             0s 3ms/step -
binary_accuracy: 0.9482 - loss: 0.1553 - val_binary_accuracy: 0.9442 - val_loss:
0.1715
Epoch 16/100
27/27             0s 3ms/step -
binary_accuracy: 0.9276 - loss: 0.1448 - val_binary_accuracy: 0.9442 - val_loss:
0.1691
Epoch 17/100
27/27             0s 4ms/step -
binary_accuracy: 0.9321 - loss: 0.1509 - val_binary_accuracy: 0.9442 - val_loss:
0.1654
Epoch 18/100
27/27             0s 4ms/step -
binary_accuracy: 0.9431 - loss: 0.1298 - val_binary_accuracy: 0.9442 - val_loss:
0.1642
Epoch 19/100
27/27             0s 4ms/step -
binary_accuracy: 0.9328 - loss: 0.1722 - val_binary_accuracy: 0.9442 - val_loss:
0.1580
Epoch 20/100
27/27             0s 4ms/step -
binary_accuracy: 0.9449 - loss: 0.1306 - val_binary_accuracy: 0.9442 - val_loss:
0.1616
Epoch 21/100
27/27             0s 4ms/step -
binary_accuracy: 0.9489 - loss: 0.1272 - val_binary_accuracy: 0.9628 - val_loss:
0.1539
Epoch 22/100
27/27             0s 4ms/step -
binary_accuracy: 0.9406 - loss: 0.1614 - val_binary_accuracy: 0.9488 - val_loss:
0.1526
Epoch 23/100
27/27             0s 4ms/step -
binary_accuracy: 0.9477 - loss: 0.1299 - val_binary_accuracy: 0.9488 - val_loss:
0.1520
Epoch 24/100
27/27             0s 4ms/step -
binary_accuracy: 0.9450 - loss: 0.1361 - val_binary_accuracy: 0.9395 - val_loss:
0.1522
Epoch 25/100
27/27             0s 4ms/step -
binary_accuracy: 0.9357 - loss: 0.1207 - val_binary_accuracy: 0.9488 - val_loss:
```

```
0.1492
Epoch 26/100
27/27            0s 4ms/step -
binary_accuracy: 0.9498 - loss: 0.1315 - val_binary_accuracy: 0.9442 - val_loss:
0.1454
Epoch 27/100
27/27            0s 4ms/step -
binary_accuracy: 0.9567 - loss: 0.1167 - val_binary_accuracy: 0.9488 - val_loss:
0.1455
Epoch 28/100
27/27            0s 3ms/step -
binary_accuracy: 0.9542 - loss: 0.1199 - val_binary_accuracy: 0.9442 - val_loss:
0.1444
Epoch 29/100
27/27            0s 3ms/step -
binary_accuracy: 0.9603 - loss: 0.1154 - val_binary_accuracy: 0.9488 - val_loss:
0.1432
Epoch 30/100
27/27            0s 4ms/step -
binary_accuracy: 0.9603 - loss: 0.1151 - val_binary_accuracy: 0.9442 - val_loss:
0.1422
Epoch 31/100
27/27            0s 3ms/step -
binary_accuracy: 0.9505 - loss: 0.1139 - val_binary_accuracy: 0.9488 - val_loss:
0.1407
Epoch 32/100
27/27            0s 3ms/step -
binary_accuracy: 0.9587 - loss: 0.1122 - val_binary_accuracy: 0.9488 - val_loss:
0.1412
Epoch 33/100
27/27            0s 3ms/step -
binary_accuracy: 0.9609 - loss: 0.0998 - val_binary_accuracy: 0.9488 - val_loss:
0.1409
Epoch 34/100
27/27            0s 4ms/step -
binary_accuracy: 0.9501 - loss: 0.1471 - val_binary_accuracy: 0.9442 - val_loss:
0.1398
Epoch 35/100
27/27            0s 4ms/step -
binary_accuracy: 0.9489 - loss: 0.1322 - val_binary_accuracy: 0.9395 - val_loss:
0.1409
Epoch 36/100
27/27            0s 3ms/step -
binary_accuracy: 0.9584 - loss: 0.1152 - val_binary_accuracy: 0.9581 - val_loss:
0.1375
Epoch 37/100
27/27            0s 3ms/step -
binary_accuracy: 0.9596 - loss: 0.1099 - val_binary_accuracy: 0.9488 - val_loss:
```

```
0.1388
Epoch 38/100
27/27            0s 3ms/step -
binary_accuracy: 0.9575 - loss: 0.0980 - val_binary_accuracy: 0.9488 - val_loss:
0.1367
Epoch 39/100
27/27            0s 3ms/step -
binary_accuracy: 0.9598 - loss: 0.1312 - val_binary_accuracy: 0.9488 - val_loss:
0.1363
Epoch 40/100
27/27            0s 3ms/step -
binary_accuracy: 0.9610 - loss: 0.1217 - val_binary_accuracy: 0.9395 - val_loss:
0.1443
Epoch 41/100
27/27            0s 4ms/step -
binary_accuracy: 0.9405 - loss: 0.1350 - val_binary_accuracy: 0.9581 - val_loss:
0.1366
Epoch 42/100
27/27            0s 3ms/step -
binary_accuracy: 0.9582 - loss: 0.1065 - val_binary_accuracy: 0.9488 - val_loss:
0.1369
Epoch 43/100
27/27            0s 4ms/step -
binary_accuracy: 0.9568 - loss: 0.1198 - val_binary_accuracy: 0.9581 - val_loss:
0.1338
Epoch 44/100
27/27            0s 4ms/step -
binary_accuracy: 0.9433 - loss: 0.1365 - val_binary_accuracy: 0.9488 - val_loss:
0.1375
Epoch 45/100
27/27            0s 4ms/step -
binary_accuracy: 0.9640 - loss: 0.1086 - val_binary_accuracy: 0.9349 - val_loss:
0.1402
Epoch 46/100
27/27            0s 3ms/step -
binary_accuracy: 0.9521 - loss: 0.1153 - val_binary_accuracy: 0.9535 - val_loss:
0.1344
Epoch 47/100
27/27            0s 3ms/step -
binary_accuracy: 0.9598 - loss: 0.0867 - val_binary_accuracy: 0.9581 - val_loss:
0.1357
Epoch 48/100
27/27            0s 4ms/step -
binary_accuracy: 0.9684 - loss: 0.1014 - val_binary_accuracy: 0.9349 - val_loss:
0.1375
Epoch 49/100
27/27            0s 3ms/step -
binary_accuracy: 0.9543 - loss: 0.1222 - val_binary_accuracy: 0.9488 - val_loss:
```

```
0.1364
Epoch 50/100
27/27            0s 4ms/step -
binary_accuracy: 0.9498 - loss: 0.1374 - val_binary_accuracy: 0.9395 - val_loss:
0.1438
Epoch 51/100
27/27            0s 3ms/step -
binary_accuracy: 0.9518 - loss: 0.1067 - val_binary_accuracy: 0.9628 - val_loss:
0.1347
Epoch 52/100
27/27            0s 4ms/step -
binary_accuracy: 0.9631 - loss: 0.0905 - val_binary_accuracy: 0.9581 - val_loss:
0.1347
Epoch 53/100
27/27            0s 4ms/step -
binary_accuracy: 0.9538 - loss: 0.1097 - val_binary_accuracy: 0.9628 - val_loss:
0.1343
Epoch 54/100
27/27            0s 3ms/step -
binary_accuracy: 0.9744 - loss: 0.0762 - val_binary_accuracy: 0.9488 - val_loss:
0.1371
Epoch 55/100
27/27            0s 3ms/step -
binary_accuracy: 0.9617 - loss: 0.1069 - val_binary_accuracy: 0.9628 - val_loss:
0.1357
Epoch 56/100
27/27            0s 4ms/step -
binary_accuracy: 0.9609 - loss: 0.1152 - val_binary_accuracy: 0.9442 - val_loss:
0.1367
Epoch 57/100
27/27            0s 3ms/step -
binary_accuracy: 0.9536 - loss: 0.1062 - val_binary_accuracy: 0.9628 - val_loss:
0.1367
Epoch 58/100
27/27            0s 3ms/step -
binary_accuracy: 0.9660 - loss: 0.0950 - val_binary_accuracy: 0.9535 - val_loss:
0.1400
Epoch 59/100
27/27            0s 3ms/step -
binary_accuracy: 0.9624 - loss: 0.0927 - val_binary_accuracy: 0.9628 - val_loss:
0.1350
Epoch 60/100
27/27            0s 3ms/step -
binary_accuracy: 0.9681 - loss: 0.0866 - val_binary_accuracy: 0.9581 - val_loss:
0.1379
Epoch 61/100
27/27            0s 4ms/step -
binary_accuracy: 0.9600 - loss: 0.1081 - val_binary_accuracy: 0.9395 - val_loss:
```

```
0.1458
Epoch 62/100
27/27            0s 4ms/step -
binary_accuracy: 0.9544 - loss: 0.1016 - val_binary_accuracy: 0.9628 - val_loss:
0.1385
Epoch 63/100
27/27            0s 5ms/step -
binary_accuracy: 0.9641 - loss: 0.1001 - val_binary_accuracy: 0.9628 - val_loss:
0.1375
Epoch 64/100
27/27            0s 3ms/step -
binary_accuracy: 0.9749 - loss: 0.0838 - val_binary_accuracy: 0.9628 - val_loss:
0.1383
Epoch 65/100
27/27            0s 4ms/step -
binary_accuracy: 0.9574 - loss: 0.1056 - val_binary_accuracy: 0.9628 - val_loss:
0.1406
Epoch 66/100
27/27            0s 4ms/step -
binary_accuracy: 0.9609 - loss: 0.1088 - val_binary_accuracy: 0.9628 - val_loss:
0.1407
Epoch 67/100
27/27            0s 4ms/step -
binary_accuracy: 0.9522 - loss: 0.1087 - val_binary_accuracy: 0.9628 - val_loss:
0.1406
Epoch 68/100
27/27            0s 4ms/step -
binary_accuracy: 0.9569 - loss: 0.1012 - val_binary_accuracy: 0.9488 - val_loss:
0.1431
Epoch 69/100
27/27            0s 3ms/step -
binary_accuracy: 0.9610 - loss: 0.1046 - val_binary_accuracy: 0.9581 - val_loss:
0.1418
Epoch 70/100
27/27            0s 5ms/step -
binary_accuracy: 0.9541 - loss: 0.1059 - val_binary_accuracy: 0.9628 - val_loss:
0.1429
Epoch 71/100
27/27            0s 4ms/step -
binary_accuracy: 0.9706 - loss: 0.0874 - val_binary_accuracy: 0.9628 - val_loss:
0.1416
Epoch 72/100
27/27            0s 4ms/step -
binary_accuracy: 0.9671 - loss: 0.1033 - val_binary_accuracy: 0.9395 - val_loss:
0.1526
Epoch 73/100
27/27            0s 3ms/step -
binary_accuracy: 0.9644 - loss: 0.0898 - val_binary_accuracy: 0.9581 - val_loss:
```

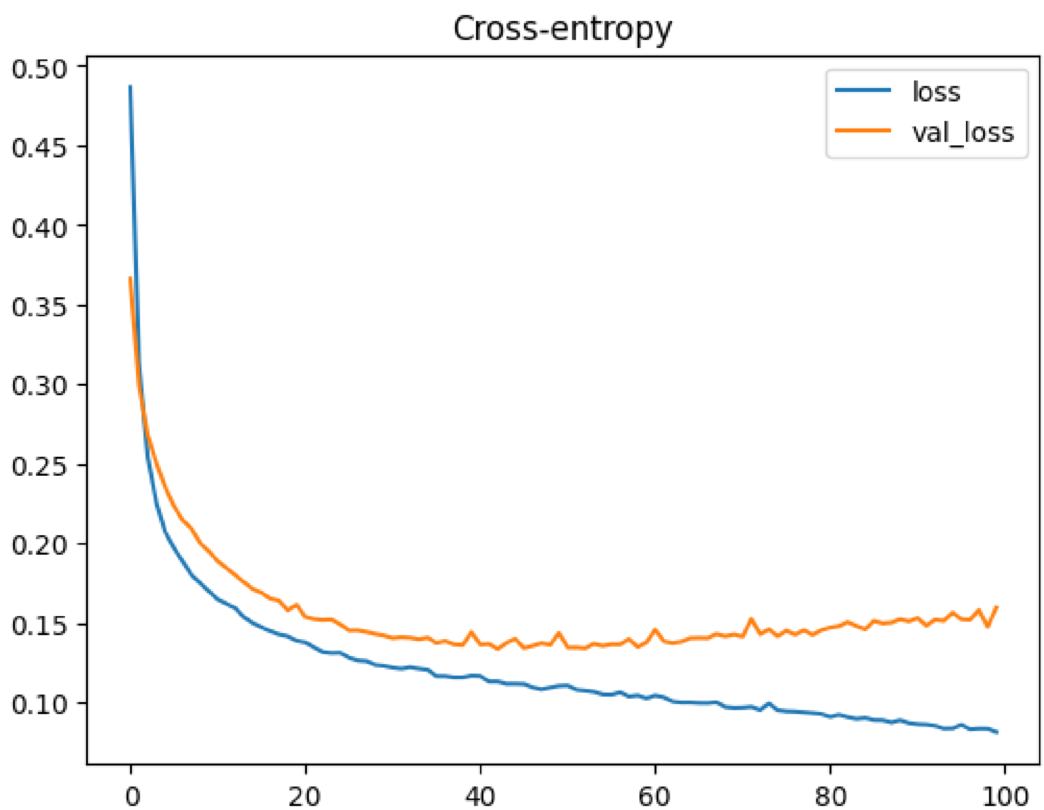
```
0.1430
Epoch 74/100
27/27              0s 4ms/step -
binary_accuracy: 0.9669 - loss: 0.0915 - val_binary_accuracy: 0.9581 - val_loss:
0.1461
Epoch 75/100
27/27              0s 4ms/step -
binary_accuracy: 0.9630 - loss: 0.1155 - val_binary_accuracy: 0.9628 - val_loss:
0.1418
Epoch 76/100
27/27              0s 3ms/step -
binary_accuracy: 0.9610 - loss: 0.1012 - val_binary_accuracy: 0.9628 - val_loss:
0.1453
Epoch 77/100
27/27              0s 4ms/step -
binary_accuracy: 0.9598 - loss: 0.1000 - val_binary_accuracy: 0.9628 - val_loss:
0.1428
Epoch 78/100
27/27              0s 4ms/step -
binary_accuracy: 0.9716 - loss: 0.0790 - val_binary_accuracy: 0.9628 - val_loss:
0.1455
Epoch 79/100
27/27              0s 3ms/step -
binary_accuracy: 0.9661 - loss: 0.0932 - val_binary_accuracy: 0.9628 - val_loss:
0.1427
Epoch 80/100
27/27              0s 4ms/step -
binary_accuracy: 0.9653 - loss: 0.0919 - val_binary_accuracy: 0.9628 - val_loss:
0.1456
Epoch 81/100
27/27              0s 3ms/step -
binary_accuracy: 0.9648 - loss: 0.0919 - val_binary_accuracy: 0.9581 - val_loss:
0.1472
Epoch 82/100
27/27              0s 4ms/step -
binary_accuracy: 0.9613 - loss: 0.1013 - val_binary_accuracy: 0.9581 - val_loss:
0.1481
Epoch 83/100
27/27              0s 3ms/step -
binary_accuracy: 0.9647 - loss: 0.0849 - val_binary_accuracy: 0.9581 - val_loss:
0.1506
Epoch 84/100
27/27              0s 3ms/step -
binary_accuracy: 0.9755 - loss: 0.0809 - val_binary_accuracy: 0.9581 - val_loss:
0.1482
Epoch 85/100
27/27              0s 3ms/step -
binary_accuracy: 0.9674 - loss: 0.0875 - val_binary_accuracy: 0.9581 - val_loss:
```

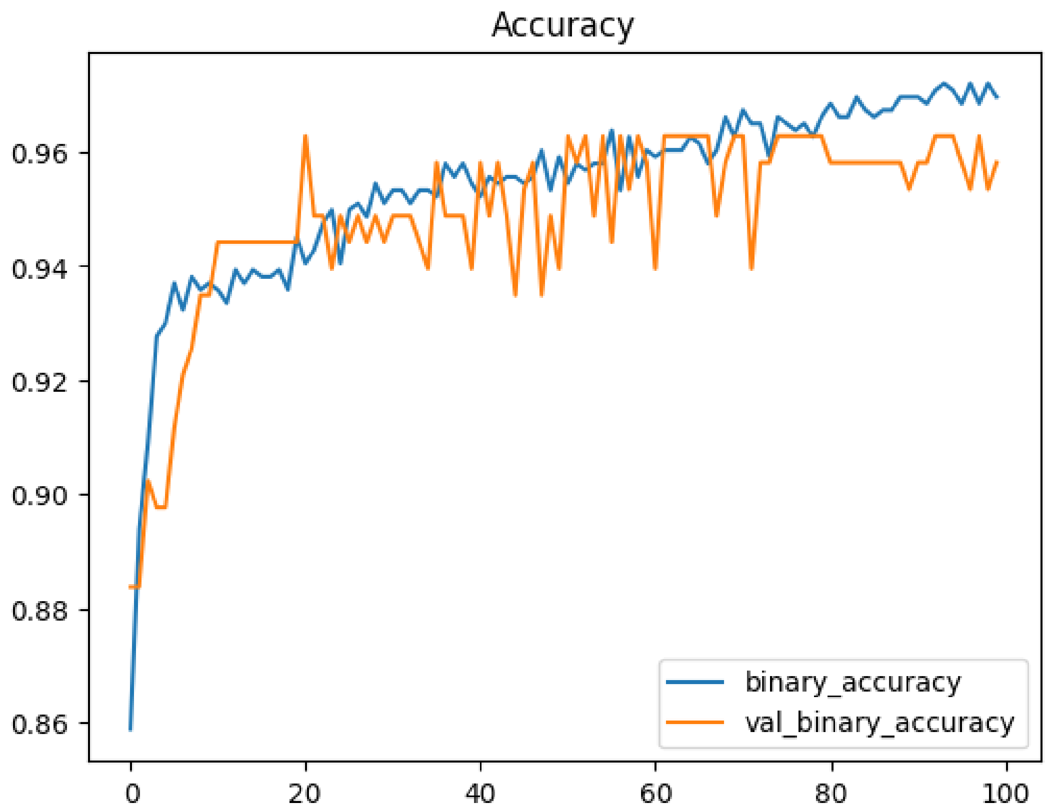
```
0.1461
Epoch 86/100
27/27              0s 3ms/step -
binary_accuracy: 0.9676 - loss: 0.0883 - val_binary_accuracy: 0.9581 - val_loss:
0.1513
Epoch 87/100
27/27              0s 4ms/step -
binary_accuracy: 0.9643 - loss: 0.0886 - val_binary_accuracy: 0.9581 - val_loss:
0.1498
Epoch 88/100
27/27              0s 3ms/step -
binary_accuracy: 0.9755 - loss: 0.0675 - val_binary_accuracy: 0.9581 - val_loss:
0.1503
Epoch 89/100
27/27              0s 4ms/step -
binary_accuracy: 0.9634 - loss: 0.1144 - val_binary_accuracy: 0.9581 - val_loss:
0.1524
Epoch 90/100
27/27              0s 4ms/step -
binary_accuracy: 0.9745 - loss: 0.0919 - val_binary_accuracy: 0.9535 - val_loss:
0.1511
Epoch 91/100
27/27              0s 3ms/step -
binary_accuracy: 0.9645 - loss: 0.0875 - val_binary_accuracy: 0.9581 - val_loss:
0.1532
Epoch 92/100
27/27              0s 3ms/step -
binary_accuracy: 0.9697 - loss: 0.0904 - val_binary_accuracy: 0.9581 - val_loss:
0.1484
Epoch 93/100
27/27              0s 3ms/step -
binary_accuracy: 0.9738 - loss: 0.0929 - val_binary_accuracy: 0.9628 - val_loss:
0.1522
Epoch 94/100
27/27              0s 3ms/step -
binary_accuracy: 0.9767 - loss: 0.0862 - val_binary_accuracy: 0.9628 - val_loss:
0.1515
Epoch 95/100
27/27              0s 4ms/step -
binary_accuracy: 0.9775 - loss: 0.0677 - val_binary_accuracy: 0.9628 - val_loss:
0.1566
Epoch 96/100
27/27              0s 4ms/step -
binary_accuracy: 0.9711 - loss: 0.0796 - val_binary_accuracy: 0.9581 - val_loss:
0.1524
Epoch 97/100
27/27              0s 5ms/step -
binary_accuracy: 0.9705 - loss: 0.0828 - val_binary_accuracy: 0.9535 - val_loss:
```

```
0.1521
Epoch 98/100
27/27              0s 4ms/step -
binary_accuracy: 0.9621 - loss: 0.0989 - val_binary_accuracy: 0.9628 - val_loss:
0.1583
Epoch 99/100
27/27              0s 4ms/step -
binary_accuracy: 0.9698 - loss: 0.0901 - val_binary_accuracy: 0.9535 - val_loss:
0.1477
Epoch 100/100
27/27              0s 4ms/step -
binary_accuracy: 0.9671 - loss: 0.0813 - val_binary_accuracy: 0.9581 - val_loss:
0.1598
```

[5]:
```python
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="Cross-entropy")
history_df.loc[:, ['binary_accuracy', 'val_binary_accuracy']].
 ↪plot(title="Accuracy")
```

[5]: <Axes: title={'center': 'Accuracy'}>





4 Make Predictions

[8]:
```
bulbasaur = np.array([45,49,49,65,65,45])
kangaskhan = np.array([105, 95, 80, 40, 80, 90])

suicune = np.array([100, 75, 115, 90, 115, 85])
zapdos = np.array([90,90,85,125,90,100])

# Combine the stats into a single array
pokemon_data = np.array([bulbasaur, kangaskhan, suicune, zapdos])

# Scale the data using the same scaler used during training
scaled_pokemon_data = scaler.transform(pokemon_data)  # Reusing the 'scaler'␣
 ↪object

# Make predictions on the scaled data
predictions = model.predict(scaled_pokemon_data)

pokemon_names = ['Bulbasaur', 'Kangaskhan', 'Suicune', 'Zapdos']
```

```
for i, prediction in enumerate(predictions):
    print(f"Confidence for {pokemon_names[i]} being legendary: {prediction}")
```

```
1/1              0s 86ms/step
Confidence for Bulbasaur being legendary: [0.00095513]
Confidence for Kangaskhan being legendary: [0.00134966]
Confidence for Suicune being legendary: [0.9147755]
Confidence for Zapdos being legendary: [0.68562794]
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:493: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

[9]:

```
lechonk = np.array([54,45,40,35,45,35])
oinkologne = np.array([110, 100, 75, 59, 80, 65])
skeledirge = np.array([104, 75, 100, 110, 75, 66])


chiyu = np.array([55,80,80,145,120,100])
koraidon = np.array([100, 135,115, 85, 110, 135])
wochien = np.array([85,90,100,100,135,75])

pokemon_test_data = np.array([lechonk, oinkologne, skeledirge, chiyu, koraidon,␣
 ↪wochien])
scaled_pokemon_test_data = scaler.transform(pokemon_test_data)
predictions = model.predict(scaled_pokemon_test_data)

pokemon_test_names = ['Lechonk', 'Oinkologne', 'Skeledirge', 'Chiyu',␣
 ↪'Koraidon', 'Wochien']

for i, prediction in enumerate(predictions):
    print(f"Confidence for {pokemon_test_names[i]} being legendary:␣
 ↪{prediction}")
```

```
1/1              0s 54ms/step
Confidence for Lechonk being legendary: [0.00674495]
Confidence for Oinkologne being legendary: [0.00015314]
Confidence for Skeledirge being legendary: [0.08187132]
Confidence for Chiyu being legendary: [0.01766323]
Confidence for Koraidon being legendary: [0.98064876]
Confidence for Wochien being legendary: [0.74192226]
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:493: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```