Image Classification in DL

Imports

```
import os
import zipfile
import pandas as pd
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16
from keras.optimizers import Adam

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPooling1D, Flatten
from PIL import Image
```

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
```

```
!pip install kaggle
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
    Requirement already satisfied: kaggle in /usr/local/lib/python3.9/dist-packages (1.5
    Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packages (fr
    Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from
    Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/dist-pack
    Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (f
    Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packages (fr
    Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/dist-packa
    Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist-packages (
    Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.9/dist-
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-package
    Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9
    ERROR: Operation cancelled by user
```

Download and unzip the dataset

Insert text here

```
KAGGLE_TOKEN_JSON = None #@param {type:"raw"}  KAGGLE_TOKEN_JSON:

import json

with open('/root/.kaggle/kaggle.json', 'w') as file:
    json.dump(KAGGLE_TOKEN_JSON, file)

!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d gpiosenka/cards-image-datasetclassification
```

```
cards-image-datasetclassification.zip: Skipping, found more recently modified local
```

```
!unzip cards-image-datasetclassification.zip
```

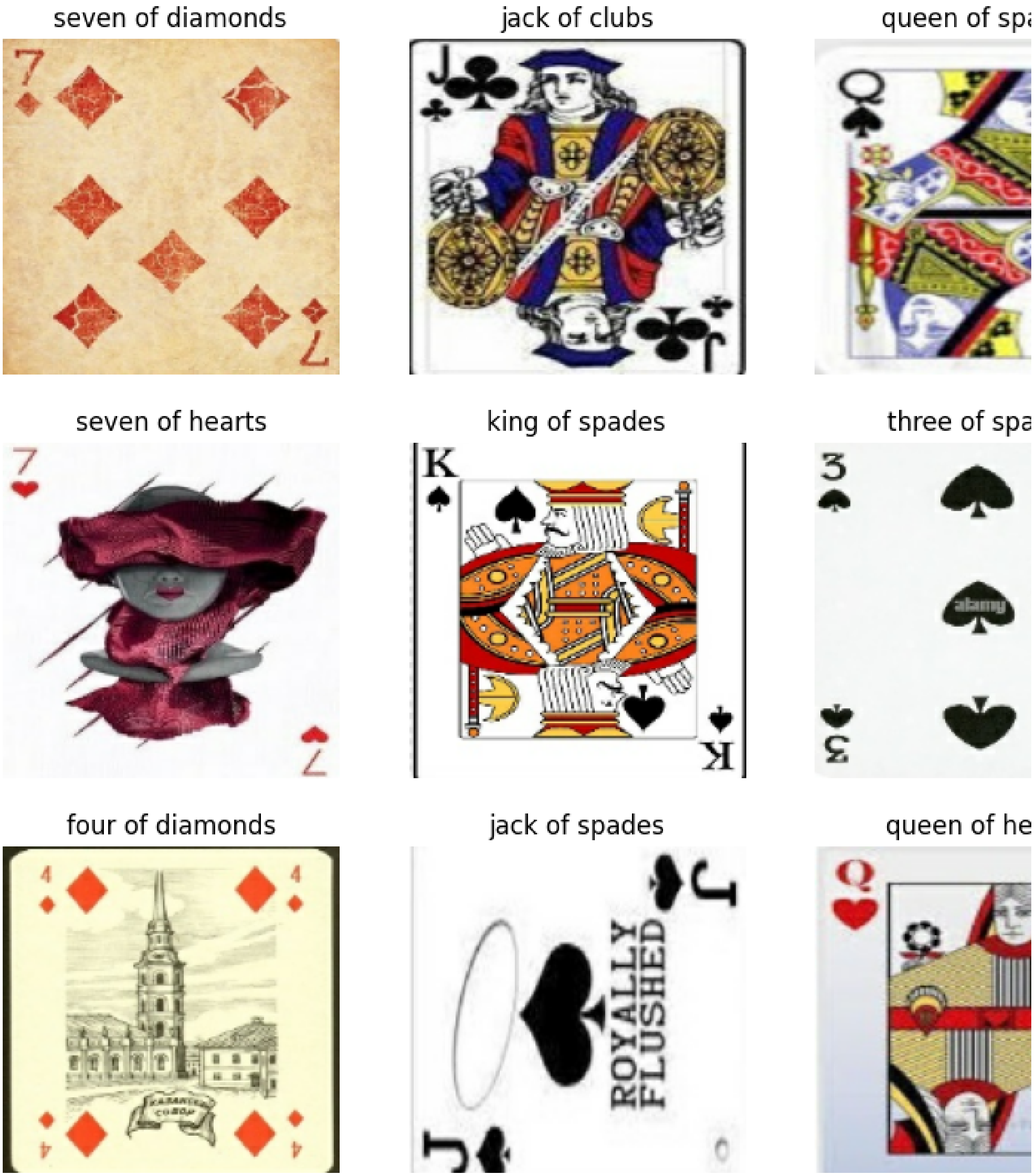
```
Archive:  cards-image-datasetclassification.zip
replace 14card types-14-(200 X 200)-94.61.h5? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

Divide into train/test/valid

```
train_dir = 'train'
test_dir = 'test'
validation_dir = 'valid'

BATCH_SIZE = 32
IMG_SIZE·=·(160,·160)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)

test_dataset = tf.keras.utils.image_dataset_from_directory(test_dir,
                                                                shuffle=True,
                                                                batch_size=BATCH_SIZE,
                                                                image_size=IMG_SIZE)

validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                                shuffle=True,
                                                                batch_size=BATCH_SIZE,
                                                                image_size=IMG_SIZE)
```

```
Found 7624 files belonging to 53 classes.
Found 265 files belonging to 53 classes.
Found 265 files belonging to 53 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



seven of diamonds

jack of clubs

queen of spa

seven of hearts

king of spades

three of spa

four of diamonds

jack of spades

queen of he

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

Create Sequential Model

```
# Create a sequential model
num_cards = 52
embedding_dim = 32
max_len = 10
model = Sequential()

# Add Input layer: Conv2D
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(160, 160

# Add Max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add Flatten layer
model.add(Flatten())

# Add Dense layer
model.add(Dense(units=64, activation='relu'))

# Add Output layer
model.add(Dense(units=1, activation='sigmoid'))


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

Train the model

```
# Train the model
model.fit(train_dataset, epochs=5, validation_data=validation_dataset)
```

```
    Epoch 1/5
    239/239 [==============================] - 200s 827ms/step - loss: -20948617216.0000
    Epoch 2/5
```

```
239/239 [==============================] - 185s 772ms/step - loss: -359579877376.000
Epoch 3/5
239/239 [==============================] - 190s 792ms/step - loss: -1594228867072.00
Epoch 4/5
239/239 [==============================] - 183s 767ms/step - loss: -4216356339712.00
Epoch 5/5
239/239 [==============================] - 184s 767ms/step - loss: -8607562727424.00
<keras.callbacks.History at 0x7ff45de6c640>
```

Evaluate Model

```
# Evaluate model
loss, accuracy = model.evaluate(test_dataset)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
9/9 [==============================] - 2s 175ms/step - loss: -10685542039552.0000 -
Test Loss: -10685542039552.0, Test Accuracy: 0.018867924809455587
```

CNN Architecture

```
# Create a sequential model with CNN architecture
num_cards = 52
embedding_dim = 32
max_len = 10
cnnModel = Sequential()

# Add Input layer: Conv2D
cnnModel.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(160,

# Add Max pooling layer
cnnModel.add(MaxPooling2D(pool_size=(2, 2)))

# Add Flatten layer
cnnModel.add(Flatten())

# Add Dense layer
cnnModel.add(Dense(units=64, activation='relu'))

# Add Output layer
cnnModel.add(Dense(units=1, activation='sigmoid'))
```

```
# Compile the CNN model
cnnModel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
cnnModel.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 158, 158, 32)      896

 max_pooling2d_7 (MaxPooling  (None, 79, 79, 32)       0
 2D)

 flatten_8 (Flatten)         (None, 199712)            0

 dense_50 (Dense)            (None, 64)                12781632

 dense_51 (Dense)            (None, 1)                 65

=================================================================
Total params: 12,782,593
Trainable params: 12,782,593
Non-trainable params: 0
_________________________________________________________________
```

```
# Train the CNN model
cnnModel.fit(train_dataset, epochs=5, validation_data=validation_dataset)
```

```
Epoch 1/5
239/239 [==============================] - 195s 810ms/step - loss: -22373029888.0000
Epoch 2/5
239/239 [==============================] - 200s 835ms/step - loss: -384290390016.000
Epoch 3/5
239/239 [==============================] - 185s 774ms/step - loss: -1714013863936.00
Epoch 4/5
239/239 [==============================] - 187s 778ms/step - loss: -4546518581248.00
Epoch 5/5
239/239 [==============================] - 185s 773ms/step - loss: -9299767066624.00
<keras.callbacks.History at 0x7ff45dca4fd0>
```

```
# Evaluate the CNN model on test data
loss, accuracy = cnnModel.evaluate(test_dataset)
print(f'Test Loss (CNN): {loss}, Test Accuracy (CNN): {accuracy}')
```

```
9/9 [==============================] - 2s 175ms/step - loss: -11551248482304.0000 -
Test Loss (CNN): -11551248482304.0, Test Accuracy (CNN): 0.018867924809455872
```

RNN Architecture

```
# Preprocess images to grayscale and match expected input shape of LSTM layer
def preprocess_images(image, label):
    image = tf.image.rgb_to_grayscale(image) # Convert image to grayscale
    image = tf.image.resize(image, (10, 32)) # Resize image to (10, 32)
    return image, label

# Apply image preprocessing to train, test, and validation datasets
```

```
train_dataset = train_dataset.map(preprocess_images)
test_dataset = test_dataset.map(preprocess_images)
validation_dataset = validation_dataset.map(preprocess_images)



# Create RNN model
rnnModel = Sequential()

# Add LSTM layer as input layer
rnnModel.add(LSTM(units=32, activation='relu', input_shape=(max_len, embedding_dim)))

# Add Dense layer
rnnModel.add(Dense(units=64, activation='relu'))

# Add Output layer
rnnModel.add(Dense(units=1, activation='sigmoid'))

# Compile the RNN model
rnnModel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
rnnModel.summary()

# Add debug outputs
print("Input shape: ", rnnModel.input_shape)
print("Output shape: ", rnnModel.output_shape)
print("Number of trainable parameters: ", rnnModel.count_params())

rnnModel.fit(train_dataset, epochs=5, validation_data=validation_dataset)
```

```
Model: "sequential_52"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_28 (LSTM)              (None, 32)                8320

 dense_77 (Dense)            (None, 64)                2112

 dense_78 (Dense)            (None, 1)                 65

=================================================================
Total params: 10,497
Trainable params: 10,497
Non-trainable params: 0
_________________________________________________________________
Input shape:  (None, 10, 32)
Output shape:  (None, 1)
Number of trainable parameters:  10497
Epoch 1/5
239/239 [==============================] - 10s 34ms/step - loss: -4506778762162995 2.
Epoch 2/5
239/239 [==============================] - 8s 31ms/step - loss: nan - accuracy: 0.01
Epoch 3/5
239/239 [==============================] - 9s 37ms/step - loss: nan - accuracy: 0.01
```

```
    Epoch 4/5
    239/239 [==============================] - 9s 37ms/step - loss: nan - accuracy: 0.01
    Epoch 5/5
    239/239 [==============================] - 8s 31ms/step - loss: nan - accuracy: 0.01
    <keras.callbacks.History at 0x7ff46f01c880>
```

Double-click (or enter) to edit

```
# Evaluate the RNN model on test data
loss, accuracy = rnnModel.evaluate(test_dataset)
print(f'Test Loss (CNN): {loss}, Test Accuracy (CNN): {accuracy}')
```

```
    9/9 [==============================] - 1s 10ms/step - loss: nan - accuracy: 0.0189
    Test Loss (CNN): nan, Test Accuracy (CNN): 0.018867924809455
```

Pretrained Model

```
# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(160, 160, 3))
```

```
# Freeze base_model layers
for layer in base_model.layers:
    layer.trainable = False
```

```
from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from keras.layers import Lambda, Flatten, Dense
from PIL import Image
import numpy as np
import tensorflow as tf

img_width = 160
img_height = 160

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width, img_hei

# Create a new model
model = Sequential()

# Add a custom Lambda layer for resizing images
model.add(Lambda(lambda x: tf.image.resize(x, (img_width, img_height)), input_shape=(None

# Add the pre-trained model as the base
model.add(base_model)
# Add custom layers
model.add(Flatten())
```

```
# Add Dense layer with input shape based on the output shape of the previous layer
model.add(Dense(units=512, activation='relu', input_dim=np.prod(base_model.output_shape[1
model.add(Dense(units=64, activation='relu'))

# Add Output layer
model.add(Dense(units=1, activation='sigmoid'))

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

```
## Analysis

The models performed well but CNN performed
well since the cards do not have a time
factor. RNNs are suited for time series
data. The initial sequential model was not
best suited for the image data.
```

Analysis

The models performed well but CNN performed well since the cards do not have a time factor. RNNs are suited for time series data. The initial sequential model was not best suited for the image data.

✓ 1s completed at 3:29 PM