Cards Classification

About Dataset

- 53 classes
- 7624 train, 265 test, 265 validation images
- 224 X 224 X 3 jpg format

The train, test and validation directories are partitioned into 53 subdirectories, one for each of the 53 types of cards. The dataset also includes a csv file which can be used to load the datasets.

In [1]:
```
import os

datasetDirectory = "dataset"
trainDirectory = datasetDirectory + "/train"
testDirectory = datasetDirectory + "/test"
validationDirectory = datasetDirectory + "/valid"

# Print all the directory name in the train directory
classesLabel = os.listdir(trainDirectory)
classesNumber = len(classesLabel)
print("Classes number: ", classesNumber)
print("Classes: ", classesLabel)
```

```
Classes number:  53
Classes:  ['ace of clubs', 'ace of diamonds', 'ace of hearts', 'ace of spades', 'eight of clubs', 'eight of diam
onds', 'eight of hearts', 'eight of spades', 'five of clubs', 'five of diamonds', 'five of hearts', 'five of spa
des', 'four of clubs', 'four of diamonds', 'four of hearts', 'four of spades', 'jack of clubs', 'jack of diamond
s', 'jack of hearts', 'jack of spades', 'joker', 'king of clubs', 'king of diamonds', 'king of hearts', 'king of
spades', 'nine of clubs', 'nine of diamonds', 'nine of hearts', 'nine of spades', 'queen of clubs', 'queen of di
amonds', 'queen of hearts', 'queen of spades', 'seven of clubs', 'seven of diamonds', 'seven of hearts', 'seven
of spades', 'six of clubs', 'six of diamonds', 'six of hearts', 'six of spades', 'ten of clubs', 'ten of diamond
s', 'ten of hearts', 'ten of spades', 'three of clubs', 'three of diamonds', 'three of hearts', 'three of spade
s', 'two of clubs', 'two of diamonds', 'two of hearts', 'two of spades']
```

Data Visualization

In [2]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random

def getRandomImage(target_dir, target_class):
    target_folder = target_dir + '/'+ target_class

    random_image = random.sample(os.listdir(target_folder), 1)# Get a random image path

    return mpimg.imread(target_folder + "/" + random_image[0])


target_class = random.choice(classesLabel)
img = getRandomImage(target_dir=trainDirectory,
                     target_class=target_class)

print(f"Image shape: {img.shape}") # show the shape of the image

plt.imshow(img)
plt.title(target_class)
plt.axis("off");
```

```
Image shape: (224, 224, 3)
```

ace of spades

In [3]:
```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Data augmentation and normalization for training
transformTrain = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    #transforms.RandomHorizontalFlip(),
    #transforms.RandomRotation(10),
    #transforms.Normalize(mean = [0.5, 0.5, 0.5], std = [0.5, 0.5, 0.5])
])

# Just normalization for validation
transformValidationTest = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    #transforms.Normalize(mean = [0.5, 0.5, 0.5], std = [0.5, 0.5, 0.5])
])

# Import the datasets from directory and apply the transforms
trainData = datasets.ImageFolder(trainDirectory, transform=transformTrain)
validData = datasets.ImageFolder(validationDirectory, transform=transformValidationTest)
testData = datasets.ImageFolder(testDirectory, transform=transformValidationTest)


# Define the dataloaders
batch_size = 32
train_dataloader = DataLoader(trainData, batch_size=batch_size, shuffle=True)
valid_dataloader = DataLoader(validData, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(testData, batch_size=batch_size, shuffle=False)

print(device)
```

```
cpu
```

In [4]:
```
import torch.nn as nn
import torch.optim as optim

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=0)
        self.activationFunction1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3)
        self.activationFunction2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3)
        self.activationFunction3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 26 * 26, 128)
```

```
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(128, 128)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(128, classesNumber)
        #self.sf = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.activationFunction1(self.conv1(x))
        out = self.pool1(out)
        out = self.activationFunction2(self.conv2(out))
        out = self.pool2(out)
        out = self.activationFunction3(self.conv3(out))
        out = self.pool3(out)

        out = self.flatten(out)
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        #out = self.sf(out)
        return out

model = ConvNet().to(device)
```

In [4]:
```
criterion = nn.CrossEntropyLoss()#Implement already softmax
optimizer = optim.Adadelta(model.parameters(), lr=1)

from torchsummary import summary
summary(model, (3, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 222, 222]           1,792
              ReLU-2         [-1, 64, 222, 222]               0
         MaxPool2d-3         [-1, 64, 111, 111]               0
            Conv2d-4         [-1, 64, 109, 109]          36,928
              ReLU-5         [-1, 64, 109, 109]               0
         MaxPool2d-6           [-1, 64, 54, 54]               0
            Conv2d-7           [-1, 64, 52, 52]          36,928
              ReLU-8           [-1, 64, 52, 52]               0
         MaxPool2d-9           [-1, 64, 26, 26]               0
           Flatten-10                [-1, 43264]               0
            Linear-11                  [-1, 128]       5,537,920
              ReLU-12                  [-1, 128]               0
            Linear-13                  [-1, 128]          16,512
              ReLU-14                  [-1, 128]               0
            Linear-15                   [-1, 53]           6,837
================================================================
Total params: 5,636,917
Trainable params: 5,636,917
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 70.48
Params size (MB): 21.50
Estimated Total Size (MB): 92.55
----------------------------------------------------------------
```

In [5]:
```
import numpy as np

epochs = 10

training_losses = []
training_accuracies = []
validation_losses = []
validation_accuracies = []

min_validation_loss = np.inf

for epoch in range(epochs):
    batch_loss = 0.0
    train_loss = 0.0
    total = 0
    correct = 0
    model.train()
    for i, (inputs, labels) in enumerate(train_dataloader, 1):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
```

```
        optimizer.step()
        train_loss += loss.item()
        batch_loss += loss.item()
        if i % 10 == 0:
            print(f'Epoch {epoch+1}, Batch {i}, Loss: {batch_loss/10:.5f}')
            batch_loss = 0.0

        # Get the accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    training_losses.append(train_loss/len(train_dataloader))
    training_accuracies.append(100 * correct / total)

    # Validation
    validation_loss = 0.0
    total = 0
    correct = 0
    model.eval()
    for inputs, labels in valid_dataloader:
        # Get the loss
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        validation_loss += loss.item()
        # Get the accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    validation_losses.append(validation_loss/len(valid_dataloader))
    validation_accuracies.append(100 * correct / total)

    print(f'Epoch {epoch+1}\nTraining:\nLoss: {training_losses[-1]} Accuracy: {training_accuracies[-1]}\nValidati

    if min_validation_loss > validation_losses[-1]:
        print(f'Validation Loss Decreased({min_validation_loss:.6f}--->{validation_losses[-1]:.6f})\tSaving The M
        min_validation_loss = validation_losses[-1]

        torch.save(model.state_dict(), 'model.pth')

print('Finished Training')
```

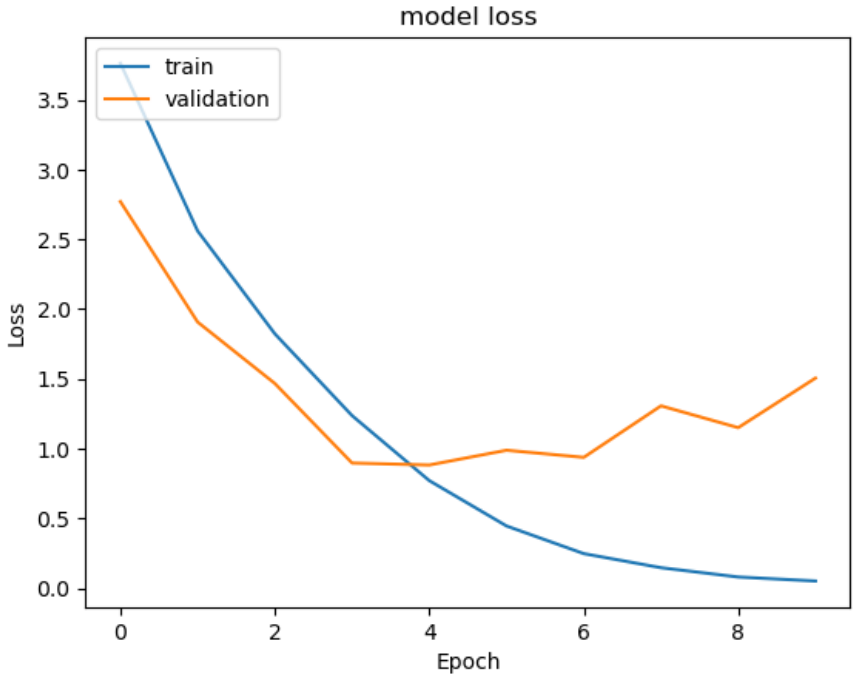
```
Epoch 1, Batch 10, Loss: 4.02831
Epoch 1, Batch 20, Loss: 3.96996
Epoch 1, Batch 30, Loss: 3.96843
Epoch 1, Batch 40, Loss: 3.96241
Epoch 1, Batch 50, Loss: 3.98138
Epoch 1, Batch 60, Loss: 3.97281
Epoch 1, Batch 70, Loss: 3.96235
Epoch 1, Batch 80, Loss: 3.97403
Epoch 1, Batch 90, Loss: 3.97237
Epoch 1, Batch 100, Loss: 3.95510
Epoch 1, Batch 110, Loss: 3.99911
Epoch 1, Batch 120, Loss: 3.96664
Epoch 1, Batch 130, Loss: 3.95791
Epoch 1, Batch 140, Loss: 3.92697
Epoch 1, Batch 150, Loss: 3.93861
Epoch 1, Batch 160, Loss: 3.89220
Epoch 1, Batch 170, Loss: 3.63345
Epoch 1, Batch 180, Loss: 3.54239
Epoch 1, Batch 190, Loss: 3.40278
Epoch 1, Batch 200, Loss: 3.26670
Epoch 1, Batch 210, Loss: 3.27601
Epoch 1, Batch 220, Loss: 3.42890
Epoch 1, Batch 230, Loss: 3.15860
Epoch 1
Training:
Loss: 3.7617159997070186 Accuracy: 5.430220356768101
Validation:
Loss: 2.7707922988467746 Accuracy: 15.09433962264151
Validation Loss Decreased(inf--->2.770792)      Saving The Model
Epoch 2, Batch 10, Loss: 2.94614
Epoch 2, Batch 20, Loss: 3.06021
Epoch 2, Batch 30, Loss: 2.83517
Epoch 2, Batch 40, Loss: 2.90889
Epoch 2, Batch 50, Loss: 2.92072
Epoch 2, Batch 60, Loss: 2.82512
Epoch 2, Batch 70, Loss: 2.80092
Epoch 2, Batch 80, Loss: 2.66653
Epoch 2, Batch 90, Loss: 2.69129
Epoch 2, Batch 100, Loss: 2.57824
Epoch 2, Batch 110, Loss: 2.75615
Epoch 2, Batch 120, Loss: 2.47181
Epoch 2, Batch 130, Loss: 2.60098
Epoch 2, Batch 140, Loss: 2.28105
Epoch 2, Batch 150, Loss: 2.36723
Epoch 2, Batch 160, Loss: 2.26091
Epoch 2, Batch 170, Loss: 2.46428
Epoch 2, Batch 180, Loss: 2.34826
Epoch 2, Batch 190, Loss: 2.31044
Epoch 2, Batch 200, Loss: 2.26359
Epoch 2, Batch 210, Loss: 2.37103
Epoch 2, Batch 220, Loss: 2.14048
Epoch 2, Batch 230, Loss: 2.38708
Epoch 2
Training:
Loss: 2.5604208732748632 Accuracy: 24.160545645330537
Validation:
Loss: 1.9076543384128146 Accuracy: 36.60377358490566
Validation Loss Decreased(2.770792--->1.907654) Saving The Model
Epoch 3, Batch 10, Loss: 2.05849
Epoch 3, Batch 20, Loss: 1.93658
Epoch 3, Batch 30, Loss: 1.93090
Epoch 3, Batch 40, Loss: 1.88411
Epoch 3, Batch 50, Loss: 1.99142
Epoch 3, Batch 60, Loss: 1.85839
Epoch 3, Batch 70, Loss: 1.81280
Epoch 3, Batch 80, Loss: 2.01742
Epoch 3, Batch 90, Loss: 1.84832
Epoch 3, Batch 100, Loss: 1.83959
Epoch 3, Batch 110, Loss: 1.95811
Epoch 3, Batch 120, Loss: 1.94130
Epoch 3, Batch 130, Loss: 1.82202
Epoch 3, Batch 140, Loss: 1.85737
Epoch 3, Batch 150, Loss: 1.75675
Epoch 3, Batch 160, Loss: 1.81013
Epoch 3, Batch 170, Loss: 1.69248
Epoch 3, Batch 180, Loss: 1.70702
Epoch 3, Batch 190, Loss: 1.69324
Epoch 3, Batch 200, Loss: 1.69359
Epoch 3, Batch 210, Loss: 1.80176
Epoch 3, Batch 220, Loss: 1.56019
Epoch 3, Batch 230, Loss: 1.46795
Epoch 3
Training:
```

```
Loss: 1.8233684091887215 Accuracy: 44.66159496327387
Validation:
Loss: 1.4673794243070815 Accuracy: 55.84905660377358
Validation Loss Decreased(1.907654--->1.467379) Saving The Model
Epoch 4, Batch 10, Loss: 1.33570
Epoch 4, Batch 20, Loss: 1.45383
Epoch 4, Batch 30, Loss: 1.41291
Epoch 4, Batch 40, Loss: 1.22064
Epoch 4, Batch 50, Loss: 1.41467
Epoch 4, Batch 60, Loss: 1.30695
Epoch 4, Batch 70, Loss: 1.21289
Epoch 4, Batch 80, Loss: 1.22347
Epoch 4, Batch 90, Loss: 1.35602
Epoch 4, Batch 100, Loss: 1.11160
Epoch 4, Batch 110, Loss: 1.17048
Epoch 4, Batch 120, Loss: 1.32167
Epoch 4, Batch 130, Loss: 1.31055
Epoch 4, Batch 140, Loss: 1.12854
Epoch 4, Batch 150, Loss: 1.31767
Epoch 4, Batch 160, Loss: 1.08276
Epoch 4, Batch 170, Loss: 1.22976
Epoch 4, Batch 180, Loss: 1.22610
Epoch 4, Batch 190, Loss: 1.15975
Epoch 4, Batch 200, Loss: 1.10570
Epoch 4, Batch 210, Loss: 1.14699
Epoch 4, Batch 220, Loss: 1.15364
Epoch 4, Batch 230, Loss: 1.17017
Epoch 4
Training:
Loss: 1.236899769580514 Accuracy: 63.693599160545645
Validation:
Loss: 0.897301541434394 Accuracy: 73.9622641509434
Validation Loss Decreased(1.467379--->0.897302) Saving The Model
Epoch 5, Batch 10, Loss: 0.80966
Epoch 5, Batch 20, Loss: 0.83260
Epoch 5, Batch 30, Loss: 0.65743
Epoch 5, Batch 40, Loss: 0.82106
Epoch 5, Batch 50, Loss: 0.64108
Epoch 5, Batch 60, Loss: 0.79999
Epoch 5, Batch 70, Loss: 0.76105
Epoch 5, Batch 80, Loss: 0.81981
Epoch 5, Batch 90, Loss: 0.83555
Epoch 5, Batch 100, Loss: 0.70170
Epoch 5, Batch 110, Loss: 0.67244
Epoch 5, Batch 120, Loss: 0.72099
Epoch 5, Batch 130, Loss: 1.04919
Epoch 5, Batch 140, Loss: 0.70917
Epoch 5, Batch 150, Loss: 0.75318
Epoch 5, Batch 160, Loss: 0.90965
Epoch 5, Batch 170, Loss: 0.90292
Epoch 5, Batch 180, Loss: 0.77934
Epoch 5, Batch 190, Loss: 0.66680
Epoch 5, Batch 200, Loss: 0.60652
Epoch 5, Batch 210, Loss: 0.70123
Epoch 5, Batch 220, Loss: 0.82620
Epoch 5, Batch 230, Loss: 0.76194
Epoch 5
Training:
Loss: 0.771143449899043 Accuracy: 77.03305351521512
Validation:
Loss: 0.883181995815701 Accuracy: 78.49056603773585
Validation Loss Decreased(0.897302--->0.883182) Saving The Model
Epoch 6, Batch 10, Loss: 0.43962
Epoch 6, Batch 20, Loss: 0.41439
Epoch 6, Batch 30, Loss: 0.31776
Epoch 6, Batch 40, Loss: 0.38464
Epoch 6, Batch 50, Loss: 0.39704
Epoch 6, Batch 60, Loss: 0.44621
Epoch 6, Batch 70, Loss: 0.41497
Epoch 6, Batch 80, Loss: 0.53445
Epoch 6, Batch 90, Loss: 0.53239
Epoch 6, Batch 100, Loss: 0.31206
Epoch 6, Batch 110, Loss: 0.51209
Epoch 6, Batch 120, Loss: 0.42062
Epoch 6, Batch 130, Loss: 0.53082
Epoch 6, Batch 140, Loss: 0.63159
Epoch 6, Batch 150, Loss: 0.46239
Epoch 6, Batch 160, Loss: 0.47770
Epoch 6, Batch 170, Loss: 0.37988
Epoch 6, Batch 180, Loss: 0.41853
Epoch 6, Batch 190, Loss: 0.44608
Epoch 6, Batch 200, Loss: 0.52589
Epoch 6, Batch 210, Loss: 0.42120
```

```
Epoch 6, Batch 220, Loss: 0.33590
Epoch 6, Batch 230, Loss: 0.51524
Epoch 6
Training:
Loss: 0.44615476757724415 Accuracy: 86.83105981112277
Validation:
Loss: 0.9888691604137421 Accuracy: 80.37735849056604
Epoch 7, Batch 10, Loss: 0.21108
Epoch 7, Batch 20, Loss: 0.23777
Epoch 7, Batch 30, Loss: 0.22857
Epoch 7, Batch 40, Loss: 0.22087
Epoch 7, Batch 50, Loss: 0.15627
Epoch 7, Batch 60, Loss: 0.23187
Epoch 7, Batch 70, Loss: 0.20295
Epoch 7, Batch 80, Loss: 0.27836
Epoch 7, Batch 90, Loss: 0.21386
Epoch 7, Batch 100, Loss: 0.27011
Epoch 7, Batch 110, Loss: 0.29649
Epoch 7, Batch 120, Loss: 0.27372
Epoch 7, Batch 130, Loss: 0.26053
Epoch 7, Batch 140, Loss: 0.32344
Epoch 7, Batch 150, Loss: 0.25607
Epoch 7, Batch 160, Loss: 0.20874
Epoch 7, Batch 170, Loss: 0.27619
Epoch 7, Batch 180, Loss: 0.31186
Epoch 7, Batch 190, Loss: 0.26663
Epoch 7, Batch 200, Loss: 0.17445
Epoch 7, Batch 210, Loss: 0.28508
Epoch 7, Batch 220, Loss: 0.37406
Epoch 7, Batch 230, Loss: 0.18613
Epoch 7
Training:
Loss: 0.24807609518732734 Accuracy: 92.75970619097586
Validation:
Loss: 0.9378467864460416 Accuracy: 82.64150943396227
Epoch 8, Batch 10, Loss: 0.04973
Epoch 8, Batch 20, Loss: 0.10971
Epoch 8, Batch 30, Loss: 0.25620
Epoch 8, Batch 40, Loss: 0.26518
Epoch 8, Batch 50, Loss: 0.16572
Epoch 8, Batch 60, Loss: 0.13817
Epoch 8, Batch 70, Loss: 0.14085
Epoch 8, Batch 80, Loss: 0.13331
Epoch 8, Batch 90, Loss: 0.13938
Epoch 8, Batch 100, Loss: 0.17758
Epoch 8, Batch 110, Loss: 0.15218
Epoch 8, Batch 120, Loss: 0.12360
Epoch 8, Batch 130, Loss: 0.11798
Epoch 8, Batch 140, Loss: 0.25537
Epoch 8, Batch 150, Loss: 0.13745
Epoch 8, Batch 160, Loss: 0.06896
Epoch 8, Batch 170, Loss: 0.15021
Epoch 8, Batch 180, Loss: 0.15779
Epoch 8, Batch 190, Loss: 0.11756
Epoch 8, Batch 200, Loss: 0.22757
Epoch 8, Batch 210, Loss: 0.06040
Epoch 8, Batch 220, Loss: 0.22578
Epoch 8, Batch 230, Loss: 0.11373
Epoch 8
Training:
Loss: 0.14711256958953411 Accuracy: 95.98635886673662
Validation:
Loss: 1.3070280154546101 Accuracy: 82.64150943396227
Epoch 9, Batch 10, Loss: 0.04964
Epoch 9, Batch 20, Loss: 0.02312
Epoch 9, Batch 30, Loss: 0.04098
Epoch 9, Batch 40, Loss: 0.07601
Epoch 9, Batch 50, Loss: 0.02644
Epoch 9, Batch 60, Loss: 0.04281
Epoch 9, Batch 70, Loss: 0.06875
Epoch 9, Batch 80, Loss: 0.08632
Epoch 9, Batch 90, Loss: 0.09954
Epoch 9, Batch 100, Loss: 0.16377
Epoch 9, Batch 110, Loss: 0.05376
Epoch 9, Batch 120, Loss: 0.12950
Epoch 9, Batch 130, Loss: 0.06122
Epoch 9, Batch 140, Loss: 0.07027
Epoch 9, Batch 150, Loss: 0.06475
Epoch 9, Batch 160, Loss: 0.11110
Epoch 9, Batch 170, Loss: 0.12396
Epoch 9, Batch 180, Loss: 0.12203
Epoch 9, Batch 190, Loss: 0.07784
Epoch 9, Batch 200, Loss: 0.06126
```

```
Epoch 9, Batch 210, Loss: 0.13642
Epoch 9, Batch 220, Loss: 0.07384
Epoch 9, Batch 230, Loss: 0.07099
Epoch 9
Training:
Loss: 0.08057096677913407 Accuracy: 98.00629590766002
Validation:
Loss: 1.1506764888763428 Accuracy: 82.64150943396227
Epoch 10, Batch 10, Loss: 0.02276
Epoch 10, Batch 20, Loss: 0.04833
Epoch 10, Batch 30, Loss: 0.03044
Epoch 10, Batch 40, Loss: 0.01076
Epoch 10, Batch 50, Loss: 0.03013
Epoch 10, Batch 60, Loss: 0.06417
Epoch 10, Batch 70, Loss: 0.04611
Epoch 10, Batch 80, Loss: 0.01314
Epoch 10, Batch 90, Loss: 0.04082
Epoch 10, Batch 100, Loss: 0.13347
Epoch 10, Batch 110, Loss: 0.10618
Epoch 10, Batch 120, Loss: 0.04181
Epoch 10, Batch 130, Loss: 0.08830
Epoch 10, Batch 140, Loss: 0.04851
Epoch 10, Batch 150, Loss: 0.05138
Epoch 10, Batch 160, Loss: 0.02716
Epoch 10, Batch 170, Loss: 0.03181
Epoch 10, Batch 180, Loss: 0.01750
Epoch 10, Batch 190, Loss: 0.03865
Epoch 10, Batch 200, Loss: 0.05188
Epoch 10, Batch 210, Loss: 0.06873
Epoch 10, Batch 220, Loss: 0.07088
Epoch 10, Batch 230, Loss: 0.07217
Epoch 10
Training:
Loss: 0.051795595451564994 Accuracy: 98.66211962224554
Validation:
Loss: 1.5057761271794636 Accuracy: 81.13207547169812
Finished Training
```

In [6]:
```python
# Plot loss
plt.plot(training_losses)
plt.plot(validation_losses)
plt.title('model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



In [7]:
```python
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
```

```
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy on test images: {100 * correct / total:.5f}%')
```

```
Accuracy on test images: 81.50943%
```

In [5]:
```
model.load_state_dict(torch.load('model.pth'))
model.eval()

y_pred = []
y_true = []
with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)

        y_pred.extend(predicted.tolist())
        y_true.extend(labels.tolist())
```

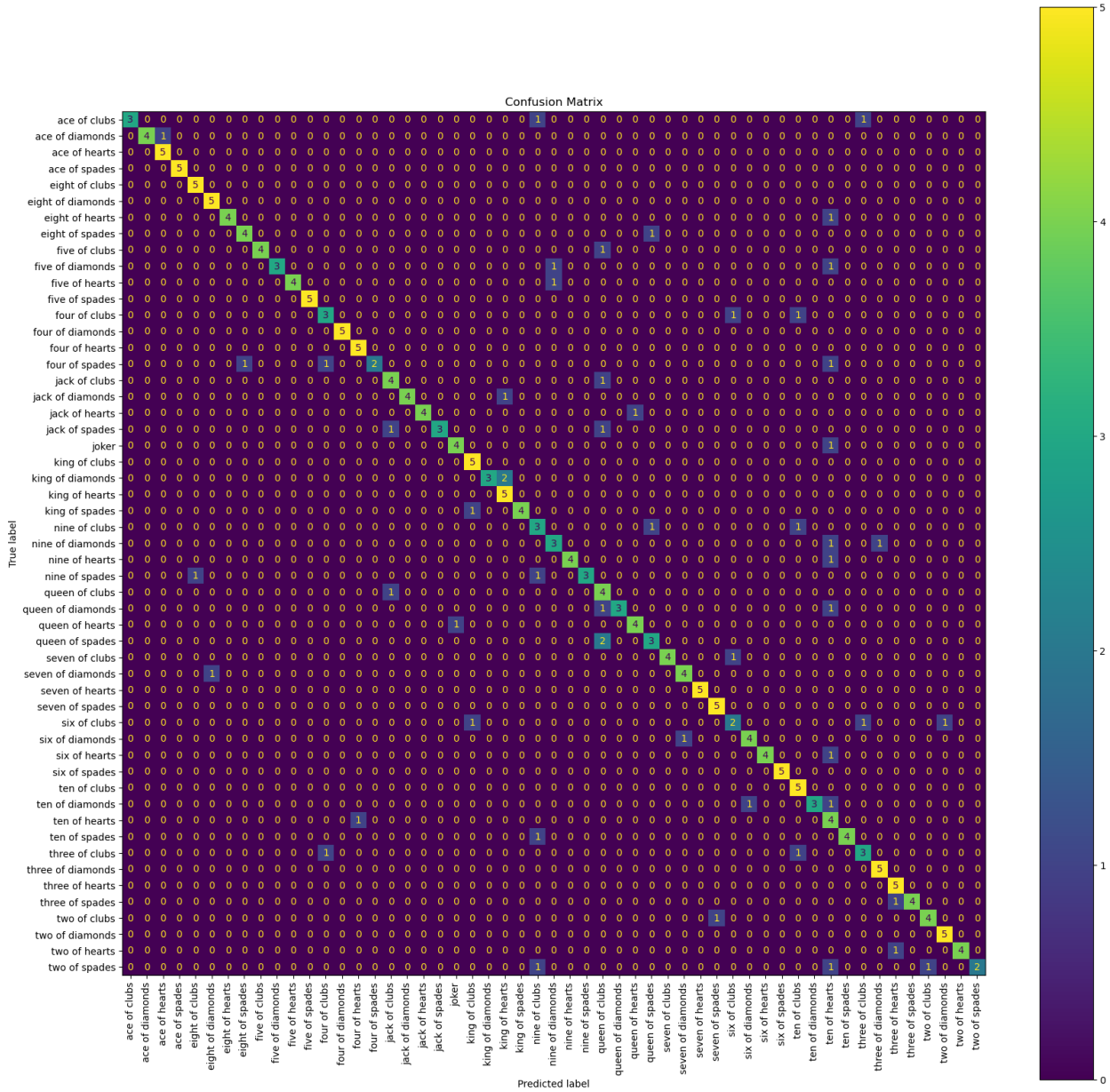
In [39]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(20, 20))

disp = ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix(y_true, y_pred),
    display_labels=classesLabel)

disp.plot(ax=ax, cmap='viridis', xticks_rotation='vertical')  # Plot on the larger axes

plt.title('Confusion Matrix')
plt.show()
```



In [7]:

```
import sklearn.metrics as metrics
import numpy as np

print('Classification Report')
print(metrics.classification_report(y_true, y_pred, target_names=classesLabel))
```

```
Classification Report
                   precision    recall  f1-score   support

     ace of clubs       1.00      0.60      0.75         5
  ace of diamonds       1.00      0.80      0.89         5
    ace of hearts       0.83      1.00      0.91         5
    ace of spades       1.00      1.00      1.00         5
   eight of clubs       0.83      1.00      0.91         5
eight of diamonds       0.83      1.00      0.91         5
  eight of hearts       1.00      0.80      0.89         5
  eight of spades       0.80      0.80      0.80         5
    five of clubs       1.00      0.80      0.89         5
 five of diamonds       1.00      0.60      0.75         5
   five of hearts       1.00      0.80      0.89         5
   five of spades       1.00      1.00      1.00         5
    four of clubs       0.60      0.60      0.60         5
 four of diamonds       1.00      1.00      1.00         5
   four of hearts       0.83      1.00      0.91         5
   four of spades       1.00      0.40      0.57         5
    jack of clubs       0.67      0.80      0.73         5
 jack of diamonds       1.00      0.80      0.89         5
   jack of hearts       1.00      0.80      0.89         5
   jack of spades       1.00      0.60      0.75         5
            joker       0.80      0.80      0.80         5
    king of clubs       0.71      1.00      0.83         5
 king of diamonds       1.00      0.60      0.75         5
   king of hearts       0.62      1.00      0.77         5
   king of spades       1.00      0.80      0.89         5
    nine of clubs       0.43      0.60      0.50         5
 nine of diamonds       0.60      0.60      0.60         5
   nine of hearts       1.00      0.80      0.89         5
   nine of spades       1.00      0.60      0.75         5
   queen of clubs       0.40      0.80      0.53         5
queen of diamonds       1.00      0.60      0.75         5
  queen of hearts       0.80      0.80      0.80         5
  queen of spades       0.60      0.60      0.60         5
   seven of clubs       1.00      0.80      0.89         5
seven of diamonds       0.80      0.80      0.80         5
  seven of hearts       1.00      1.00      1.00         5
  seven of spades       0.83      1.00      0.91         5
     six of clubs       0.50      0.40      0.44         5
  six of diamonds       0.80      0.80      0.80         5
    six of hearts       1.00      0.80      0.89         5
    six of spades       1.00      1.00      1.00         5
     ten of clubs       0.62      1.00      0.77         5
  ten of diamonds       1.00      0.60      0.75         5
    ten of hearts       0.29      0.80      0.42         5
    ten of spades       1.00      0.80      0.89         5
   three of clubs       0.60      0.60      0.60         5
three of diamonds       0.83      1.00      0.91         5
  three of hearts       0.71      1.00      0.83         5
  three of spades       1.00      0.80      0.89         5
     two of clubs       0.80      0.80      0.80         5
  two of diamonds       0.83      1.00      0.91         5
    two of hearts       1.00      0.80      0.89         5
    two of spades       1.00      0.40      0.57         5

         accuracy                           0.79       265
        macro avg       0.85      0.79      0.80       265
     weighted avg       0.85      0.79      0.80       265
```

In [34]:

```python
import matplotlib.pyplot as plt
from PIL import Image
import random

model.load_state_dict(torch.load('model.pth'))

label = random.choice(classesLabel)
image = getRandomImage(target_dir=testDirectory, target_class=label)
image = Image.fromarray(image)# Convert the NumPy array to a PIL Image

# Display the image
plt.imshow(image)
plt.title(f"Actual: {label}")
plt.axis("off")  # Turn off axis numbers and ticks
plt.show()

# Now apply the transformation and add a batch dimension
transformed_image = transformValidationTest(image).unsqueeze(0)

# Predict with the model
model.eval()  # Set the model to evaluation mode
output = model(transformed_image)
```

```
_, predicted = torch.max(output, 1)

print('Predicted:', classesLabel[predicted.item()])
```


```
Predicted: ten of hearts
```