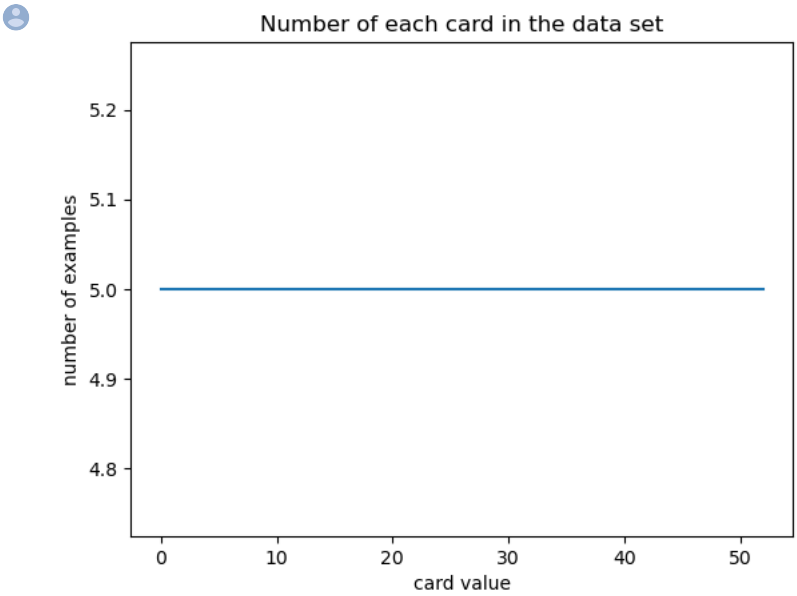
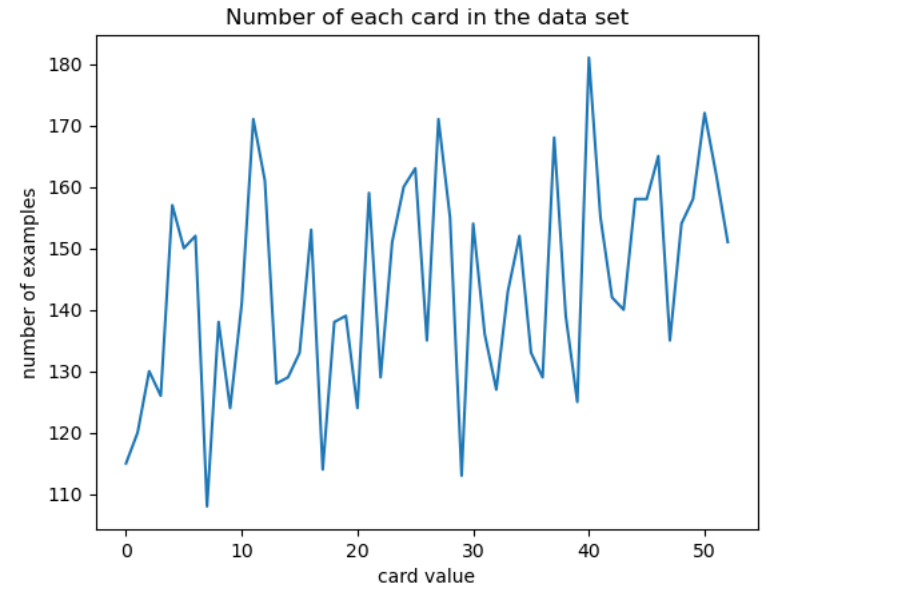
```
import numpy as np
import pandas as pd
import os
import tensorflow as tf



#create datasets
def createDataset(directoryPath):
    imagePaths = []
    values = []
    counter = [0] * 53
    h = 0

    #loop through each subdirectory
    for subdir in os.listdir(directoryPath):
        #check if the path is a directory
        if os.path.isdir(os.path.join(directoryPath, subdir)):
            # for each file in the sub directory copy the file path and value of the card
            for filename in os.listdir(directoryPath+"/"+subdir):
                parse = subdir.split(" ")
                value = 0;

                if(parse[0] == "ace"):
                    value = 1;
                elif(parse[0] == "two"):
                    value = 2;
                elif(parse[0] == "three"):
                    value = 3;
                elif(parse[0] == "four"):
                    value = 4;
                elif(parse[0] == "five"):
                    value = 5;
                elif(parse[0] == "six"):
                    value = 6;
                elif(parse[0] == "seven"):
                    value = 7;
                elif(parse[0] == "eight"):
                    value = 8;
                elif(parse[0] == "nine"):
                    value = 9;
                elif(parse[0] == "ten"):
                    value = 10;
                elif(parse[0] == "jack"):
                    value = 11;
                elif(parse[0] == "queen"):
                    value = 12;
                elif(parse[0] == "king"):
                    value = 13;

                if(parse[0] != "joker"):
                    if(parse[2] == "spades"):
                        value += 13*3;
                    elif(parse[2] == "hearts"):
                        value += 13*2;
                    elif(parse[2] == "diamonds"):
                        value += 13;

                imagePaths.append(directoryPath + "/" + subdir + "/" + filename)
                values.append(value)
                counter[value] = counter[value] + 1
                if(counter[value] > h):
                 h = counter[value]

    def load_and_preprocess_image(image_path, label):
        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [100, 100])
        image = tf.keras.applications.resnet.preprocess_input(image)
        return image, label

    dataset = tf.data.Dataset.from_tensor_slices((imagePaths, values))
    dataset = dataset.map(load_and_preprocess_image)
    plt.plot(counter)
    plt.title("Number of each card in the data set")
    plt.xlabel("card value")
```

```
    plt.ylabel("number of examples")
    plt.show()

    return dataset

testDataset = createDataset('/kaggle/input/cards-image-datasetclassification/test').batch(32)
trainDataset = createDataset('/kaggle/input/cards-image-datasetclassification/train').batch(32)



print(fullDataset)
```





<BatchDataset element_spec=(TensorSpec(shape=(None, None, 100, 100, 3), dtype=tf.float3

Distribution Graph

This graph shows the distribution of the number of cards in each data sets. A line graph was used instead of a bar graph because of how many cards their were, a bar graph caused each card's bar to be too thin to read. The cards are sorted by their value with aces being 1 and joker being 0 and each suit is 13 more then the previous suit. For example the 4 of clubs is 4 and the 4 of diamonds is 17 because the diamond is the higher suit. The suits rank as follows: clubs, diamonds, hearts and then spade. The test set is only given 5 of each card but the train set is atleast 100 for each.

```
# create the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(100, 100, 3)),
  tf.keras.layers.Dense(2048, activation="relu"),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(2048, activation="relu"),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(53, activation="softmax"),
])

model.summary()

model.compile(loss="categorical_crossentropy",
  optimizer="rmsprop",
  metrics=["accuracy"])
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_7 (Flatten)         (None, 30000)             0

 dense_17 (Dense)            (None, 2048)              61442048

 dropout_11 (Dropout)        (None, 2048)              0

 dense_18 (Dense)            (None, 2048)              4196352

 dropout_12 (Dropout)        (None, 2048)              0

 dense_19 (Dense)            (None, 53)                108597

=================================================================
Total params: 65,746,997
Trainable params: 65,746,997
Non-trainable params: 0
_________________________________________________________________
```

```
#Evaluate
history = model.fit(trainDataset,
  epochs=1,
  verbose=1)

score = model.evaluate(testDataset, verbose=0)

print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Train on 239 steps
239/239 [==============================] - 169s 661ms/step - batch: 119.0000 - size: 1.0000 - loss: 189445125532.6151 - accuracy: 0.0026
Test loss: 293939951843.55554
Test accuracy: 0.0
```

```
CNNmodel = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (9, 9), activation='relu', input_shape=(100, 100, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (9, 9), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (9, 9), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(53, activation='softmax')
])

CNNmodel.summary()

CNNmodel.compile(loss="categorical_crossentropy",
  optimizer="rmsprop",
  metrics=["accuracy"])
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 92, 92, 32)        7808

 max_pooling2d_6 (MaxPooling  (None, 46, 46, 32)       0
```

```
2D)

conv2d_7 (Conv2D)           (None, 38, 38, 64)        165952

max_pooling2d_7 (MaxPooling  (None, 19, 19, 64)       0
2D)

conv2d_8 (Conv2D)           (None, 11, 11, 128)       663680

max_pooling2d_8 (MaxPooling  (None, 5, 5, 128)        0
2D)

flatten_8 (Flatten)         (None, 3200)              0

dropout_13 (Dropout)        (None, 3200)              0

dense_20 (Dense)            (None, 53)                169653

=================================================================
Total params: 1,007,093
Trainable params: 1,007,093
Non-trainable params: 0
_________________________________________________________________
```

```
#Evaluate
history = model.fit(trainDataset,
  epochs=1,
  verbose=1)

score = model.evaluate(testDataset, verbose=0)

print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Train on 239 steps
239/239 [==============================] - 170s 665ms/step - batch: 119.0000 - size: 1.0000 - loss: 928340439691.2468 - accuracy: 0.0066
```

Model Analysis

Both of my model's preformed extremely poorly. Neither were able to get an accuracy in the double digits despite my efforts. I have tried many different changes such as batchsize, the image resize, the epoch, the number of layers, the number of nodes in each layer but none of them made too much of a difference. The hardest part of this assignment was the time it took to create each model as the calculations took so much time.

*I had some other models that did preform slightly better but I am having to redo this assignment because Kaggle didn't save my code and I have 20 minutes left as I'm writeing. They didn't perform that well so not much was really lost.

*The models are doing better now that I've reduced the batch size but I have 10 minutes left and I cant change the epoch in time.