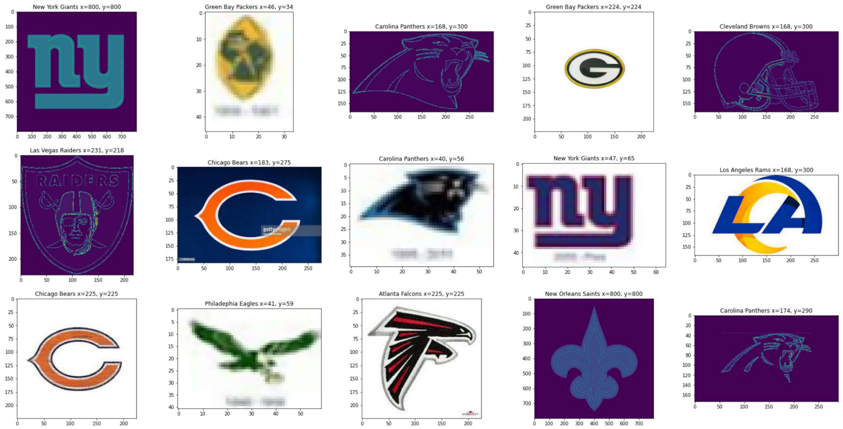
```
import numpy as np
import pandas as pd
import torch
from torch import optim
import torch.nn as nn
from PIL import Image
import os


from skimage import filters, color, morphology, io
import matplotlib.pyplot as plt
import numpy as np
import random
import skimage as si

from google.colab import auth
from google.colab import drive

import gspread

from google.auth import default
from PIL import Image

#autenticating to google
auth.authenticate_user()
creds, _ = default()
gc = gspread.authorize(creds)

drive.mount('/content/drive')
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

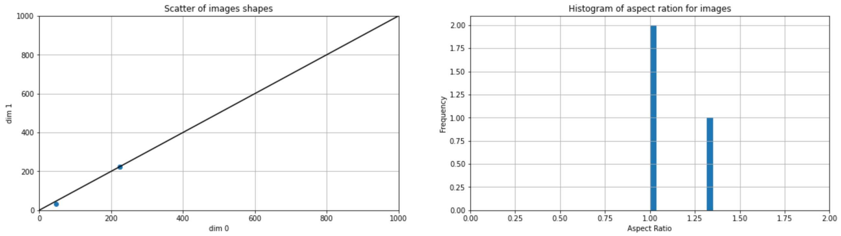
```
sh = gc.open_by_url('https://docs.google.com/spreadsheets/d/1eNgK86cm6L1OVI5iavk7mY3UQUd1VHeXgwXSTyNNuRw/edit?usp=sharing')
ws = sh.worksheet('train')

df = pd.DataFrame(ws.get_all_records())

n1 = 3
n2 = 5

fig, axs = plt.subplots(n1, n2, figsize=(32, 16))

for i in range(n1):
  for j in range(n2):
    z = random.randint(0, len(df)-1)
    pic = np.array(Image.open(df.loc[z]["filepath"]))
    shape = np.shape(pic)
    axs[i, j].imshow(pic)
    axs[i, j].set_title('%s x=%.f, y=%.f' % (df.loc[z]["team_name"], shape[0], shape[1]))
```


New York Giants x=800, y=800
Green Bay Packers x=46, y=34
Carolina Panthers x=168, y=300
Green Bay Packers x=224, y=224
Cleveland Browns x=168, y=300
Las Vegas Raiders x=231, y=218
Chicago Bears x=183, y=275
Carolina Panthers x=40, y=56
New York Giants x=47, y=65
Los Angeles Rams x=168, y=300
Chicago Bears x=225, y=225
Philadephia Eagles x=41, y=59
Atlanta Falcons x=225, y=225
New Orleans Saints x=800, y=800
Carolina Panthers x=174, y=290

```
import numpy as np

sample_df = df.sample(frac=0.01, ignore_index=True)

N = len(sample_df)
shape = np.zeros((2, N))

for i in range(N):
    tmp = np.shape(np.array(Image.open(sample_df.loc[i]["filepath"]).convert('L')))
    shape[:, i] = [tmp[0], tmp[1]]

fig, axs = plt.subplots(1, 2, figsize=(20, 5))

axs[0].scatter(shape[0, :], shape[1, :])
axs[0].plot(range(0, 1000), range(0, 1000), 'k')
```

```
axs[0].set_ylabel('dim 1')
axs[0].set_xlabel('dim 0')
axs[0].grid()
axs[0].set_title('Scatter of images shapes')
axs[0].set_xlim([0, 1000])
axs[0].set_ylim([0, 1000])

axs[1].hist(shape[0, :]/shape[1, :])
axs[1].grid()
axs[1].set_xlabel('Aspect Ratio')
axs[1].set_ylabel('Frequency')
axs[1].set_title('Histogram of aspect ration for images')
axs[1].set_xlim([0, 2])

print("Average height " + str(sum(shape[1, :]) / len(shape[1, :])))
```

```
Average height 161.0
```



```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
  def __init__(self, X, y, BatchSize, transform):
    super().__init__()
    self.BatchSize = BatchSize
    self.y = y
    self.X = X
    self.transform = transform

  def num_of_batches(self):
    """
    Detect the total number of batches
    """
    return math.floor(len(self.list_IDs) / self.BatchSize)

  def __getitem__(self,idx):
    class_id = self.y[idx]
    img = Image.open(self.X[idx])
    img = img.convert("RGBA").convert("RGB")
    img = self.transform(img)
    return img, torch.tensor(int(class_id))
  def __len__(self):
    return len(self.X)
```

```
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
from torchvision import transforms

# Shuffle dataframe
df = df.sample(frac=1)

X = df.iloc[:,0]
y = df.iloc[:,2]

transform = transforms.Compose([
                transforms.Resize([256,256]),
                transforms.RandomRotation(20, fill=256),
                transforms.ToTensor(),
                transforms.RandomAffine(degrees=0, translate=(0.025, 0.025), fill=256),
                transforms.Normalize([0.5], [0.5])
            ])

test_transform = transforms.Compose([
                transforms.Resize([256,256]),
                transforms.ToTensor(),
                transforms.Normalize((0.5,), (0.5,)),
            ])

train_ratio = 0.80
validation_ratio = 0.1
test_ratio = 0.1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_ratio, stratify = y, random_state = 0)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio), random_state = 0)

dataset_stages = ['train', 'val', 'test']

batch_size = 32
image_datasets = {'train' : CustomDataset(X_train.values, y_train.values, batch_size, transform), 'val' : CustomDataset(X_val.values, y_val.values, batch_size, test_transform), 'test' : Cu
dataloaders = {x: DataLoader(image_datasets[x], batch_size=image_datasets[x].BatchSize,
                                            shuffle=True, num_workers=0)
            for x in dataset_stages}

dataset_sizes = {x: len(image_datasets[x]) for x in dataset_stages}

print(dataset_sizes)
```

```
     {'train': 256, 'val': 32, 'test': 32}
```

```
nparray = image_datasets['train'][12][0].cpu().numpy()
image = transforms.ToPILImage()(image_datasets['train'][12][0].cpu()).convert("RGB")
display(image)
```

```
import time

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

def train_model(model, criterion, optimizer, scheduler, num_epochs=15):
    since = time.time()
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0
            num_batches = 0
            outputs = None
            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                # Loading Bar
                if (phase == 'train'):
                    num_batches += 1
                    percentage_complete = ((num_batches * batch_size) / (dataset_sizes[phase])) * 100
                    percentage_complete = np.clip(percentage_complete, 0, 100)
                    print("{:0.2f}".format(percentage_complete), "% complete", end="\r")

                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs.float(), labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        # TODO: try removal
                        torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)

                predicted = torch.max(outputs.data, 1)[1]
                running_correct = (predicted == labels).sum()
                running_corrects += running_correct
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]

            epoch_acc = running_corrects / dataset_sizes[phase]
            #epoch_acc = sum(epoch_acc) / len(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc.item()))

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    return model
```

```
from torchvision import models
from torch.optim import lr_scheduler

model_ft = models.squeezenet1_1(pretrained=True)
model_ft.num_classes = 32 #this is am important compoent of the output model
model_ft.classifier._modules["1"] = nn.Conv2d(512, model_ft.num_classes, kernel_size=(1, 1))
for param in model_ft.parameters():
    param.requires_grad = False
for param in model_ft.classifier.parameters():
    param.requires_grad = True

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.01)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_ft = train_model(model_ft.to(device), criterion, optimizer_ft, exp_lr_scheduler, 15)
```

```
Epoch 1/14
----------
train Loss: 195.6700 Acc: 0.0312
val Loss: 8.3117 Acc: 0.2188
Epoch 2/14
----------
train Loss: 128.1705 Acc: 0.0352
val Loss: 6.5645 Acc: 0.2500
```

```
Epoch 5/14
----------
train Loss: 16.0656 Acc: 0.1250
val Loss: 3.7918 Acc: 0.3750
Epoch 6/14
----------
train Loss: 3.4933 Acc: 0.4219
val Loss: 2.6368 Acc: 0.4688
Epoch 7/14
----------
train Loss: 1.8375 Acc: 0.5977
val Loss: 2.3684 Acc: 0.5312
Epoch 8/14
----------
train Loss: 1.8471 Acc: 0.6094
val Loss: 2.0991 Acc: 0.5312
Epoch 9/14
----------
train Loss: 1.6187 Acc: 0.6328
val Loss: 1.9921 Acc: 0.5000
Epoch 10/14
----------
train Loss: 1.5550 Acc: 0.6484
val Loss: 1.9330 Acc: 0.5000
Epoch 11/14
----------
train Loss: 1.3840 Acc: 0.6797
val Loss: 1.8805 Acc: 0.5312
Epoch 12/14
----------
train Loss: 1.4538 Acc: 0.6797
val Loss: 1.8304 Acc: 0.5312
Epoch 13/14
----------
train Loss: 1.3677 Acc: 0.7148
val Loss: 1.7130 Acc: 0.5625
Epoch 14/14
----------
train Loss: 1.2909 Acc: 0.7578
val Loss: 1.7002 Acc: 0.5312
```

```python
from sklearn.metrics import accuracy_score

accuracy_scores = []

running_corrects = 0
outputs = None
for inputs, labels in dataloaders['test']:
    model_ft.eval()

    # print(labels.numpy())
    inputs = inputs.to(device)
    labels = labels.to(device)

    outputs = model_ft(inputs)

    predicted = torch.max(outputs.data, 1)[1]
    running_correct = (predicted == labels).sum()
    running_corrects += running_correct

accuracy = running_corrects / dataset_sizes['test']
```

```
print("Accuracy: " + str(accuracy.item()))

     Accuracy: 0.625
```

Colab paid products - Cancel contracts here

✓ 1s completed at 4:04 PM