# Transfer Learning with monkey species classification

In [1]:
```
#import packages
import numpy as np
import pandas as pd
import torch
from torch import nn, optim
import torchvision
from torchvision import datasets, transforms, utils, models
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
from PIL import Image
import os

plt.style.use('ggplot')
plt.rcParams.update({'font.size': 14, 'axes.labelweight': 'bold', 'axes.grid': False})
```

In [2]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device.type}")
```

In [3]:
```
# Set up data
TRAIN_DIR = "../input/10-monkey-species/training/training"
VALID_DIR = "../input/10-monkey-species/validation/validation"


IMAGE_SIZE = 200
BATCH_SIZE = 64

data_transforms = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.ImageFolder(root=TRAIN_DIR, transform=data_transforms)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=T

valid_dataset = datasets.ImageFolder(root=VALID_DIR, transform=data_transforms)
valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=BATCH_SIZE, shuffle=T

# Plot samples
sample_batch = next(iter(train_loader))
plt.figure(figsize=(10, 8)); plt.axis("off"); plt.title("Sample Training Images")
plt.imshow(np.transpose(utils.make_grid(sample_batch[0], padding=1, normalize=True),(1, 2,
```

In [4]:
```
#define class labels
#class_labels = [ f.path.split('/')[-1] for f in os.scandir(TRAIN_DIR) if f.is_dir() ]
class_labels = ['alouattapalliata', 'erythrocebuspatas', 'cacajaocalvus',
                'macacafuscata', 'cebuellapygmea', 'cebuscapucinus',
                'micoargentatus', 'saimirisciureus', 'aotusnigriceps', 'trachypithecusjohr

#get a random batch of 64 iamge
image, label = next(iter(train_loader))
#choose one image at random
i = np.random.randint(0, 64)
image_con = image[i, :]
image_con = image_con.swapaxes(0, 1)
```

```
image_con = image_con.swapaxes(1, 2)
#show the image with label
plt.imshow(image_con)
plt.title(class_labels[label[i]]);
```

In [5]:

```
# Define model
def conv_block(input_channels, output_channels):
    return nn.Sequential(
        nn.Conv2d(input_channels, output_channels, 3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d((3, 3))
    )

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.main = torch.nn.Sequential(
            conv_block(3, 64),
            conv_block(64, 32),
            conv_block(32, 16),

            nn.Flatten(),
            nn.Linear(784, 300),
            nn.ReLU(),
            nn.Linear(300, 50),
            nn.ReLU(),
            nn.Linear(50, 10)
        )

    def forward(self, x):
        out = self.main(x)
#         print(out.size())
        return out


def trainer(model, criterion, optimizer, train_loader, valid_loader, epochs=20, verbose=Tr
    """Simple training wrapper for PyTorch network."""

    train_loss, valid_loss, train_accuracy, valid_accuracy = [], [], [], []
    for epoch in range(epochs):  # for each epoch
        train_batch_loss = 0
        train_batch_acc = 0
        valid_batch_loss = 0
        valid_batch_acc = 0

        # Training
        model.train()
        for X, y in train_loader:
            X, y = X.to(device), y.to(device)
            optimizer.zero_grad()
            y_hat = model(X)
            _, y_hat_labels = torch.softmax(y_hat, dim=1).topk(1, dim=1)
            loss = criterion(y_hat, y)
            loss.backward()
            optimizer.step()
            train_batch_loss += loss.item()
            train_batch_acc += (y_hat_labels.squeeze() == y).type(torch.float32).mean().it
        train_loss.append(train_batch_loss / len(train_loader))
        train_accuracy.append(train_batch_acc / len(train_loader))

        # Validation
        model.eval()
        with torch.no_grad():  # this stops pytorch doing computational graph stuff under-
            for X, y in valid_loader:
```

```
                X, y = X.to(device), y.to(device)
                y_hat = model(X)
                _, y_hat_labels = torch.softmax(y_hat, dim=1).topk(1, dim=1)
                loss = criterion(y_hat, y)
                valid_batch_loss += loss.item()
                valid_batch_acc += (y_hat_labels.squeeze() == y).type(torch.float32).mean
        valid_loss.append(valid_batch_loss / len(valid_loader))
        valid_accuracy.append(valid_batch_acc / len(valid_loader))

        # Print progress
        if verbose:
            print(f"Epoch {epoch + 1}:",
                  f"Train Loss: {train_loss[-1]:.3f}",
                  f"Train Accuracy: {train_accuracy[-1]:.2f}",
                  f"Valid Loss: {valid_loss[-1]:.3f}.",
                  f"Valid Accuracy: {valid_accuracy[-1]:.2f}")

    results = {"train_loss": train_loss,
               "train_accuracy": train_accuracy,
               "valid_loss": valid_loss,
               "valid_accuracy": valid_accuracy}
    return results
```

In [6]:

```
model = CNN()
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.002)
results = trainer(model, criterion, optimizer, train_loader, valid_loader, epochs=10)
```

In [7]:

```
#get the predictions
def plot_prediction(image, label, predictions):
    """Plot network predictions with matplotlib."""
    fig, (ax1, ax2) = plt.subplots(figsize=(8, 4), ncols=2)  # Plot
    i = np.random.randint(0, 64)
    img = image[i, :]
    img = img.swapaxes(0, 1)
    img = img.swapaxes(1, 2)
    img = img.detach().cpu().numpy()
    ax1.imshow(img)
    ax1.axis('off')
    ax1.set_title(class_labels[label[i]])
    ax2.barh(np.arange(10), predictions[i,:].detach().cpu().numpy().squeeze())
    ax2.set_title("Predictions")
    ax2.set_yticks(np.arange(10))
    ax2.set_yticklabels(class_labels)
    ax2.set_xlim(0, 1)
    plt.tight_layout();
```

In [8]:

```
# Test model on training images
image, label = next(iter(train_loader))        # Get a random batch of 64 images
image, label = image.to(device), label.to(device)
predictions = model(image)      # Get first image, flatten to shape (1, 784) and predict it
predictions = nn.Softmax(dim=1)(predictions)
plot_prediction(image, label, predictions)
```

## Feature Extractor

In [9]:

```
#Use pretrained model densenet with layer on top of it
densenet = models.densenet121(pretrained=True)
```

```
for param in densenet.parameters(): # Freeze parameters
    param.requires_grad = False

# Customize classification layers
new_layers = nn.Sequential(
    nn.Linear(1024, 50),
    nn.ReLU(),
    nn.Linear(50, 10)
)

densenet.classifier = new_layers

# Time to train
densenet.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(densenet.parameters(), lr=1e-3)
results = trainer(densenet, criterion, optimizer, train_loader, valid_loader, epochs=10)
```

## Fine tuning

In [10]:

```
densenet = models.densenet121(pretrained=True)

for param in densenet.parameters():
    param.requires_grad = False

# Unfreeze denseblock4
for param in densenet.features.denseblock4.parameters():
    param.requires_grad = True

# Customize classification layers
new_layers = nn.Sequential(
    nn.Linear(1024, 50),
    nn.ReLU(),
    nn.Linear(50, 10)
)

densenet.classifier = new_layers

# Time to train
densenet.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(densenet.parameters(), lr=1e-3)
results = trainer(densenet, criterion, optimizer, train_loader, valid_loader, epochs=10)
```

In [ ]:

```
```