Hands-On Exercise 1: Image Classification

Julia Owen, PhD
University of Washington

Setting up today's classification problem

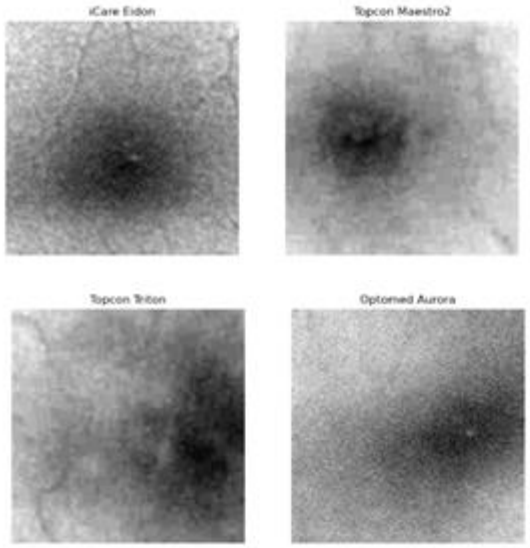
Classifying images using DL is an important area of clinical application in ophthalmology

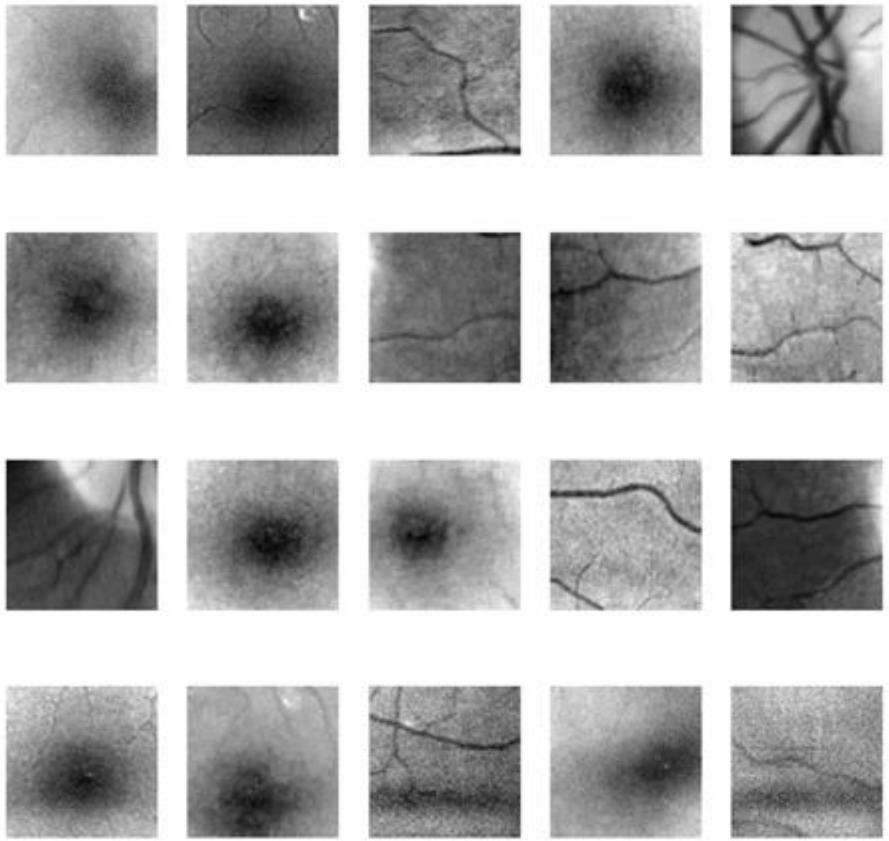
Today we will explore a toy DL problem, but everything we learn can be applied to real-world challenges

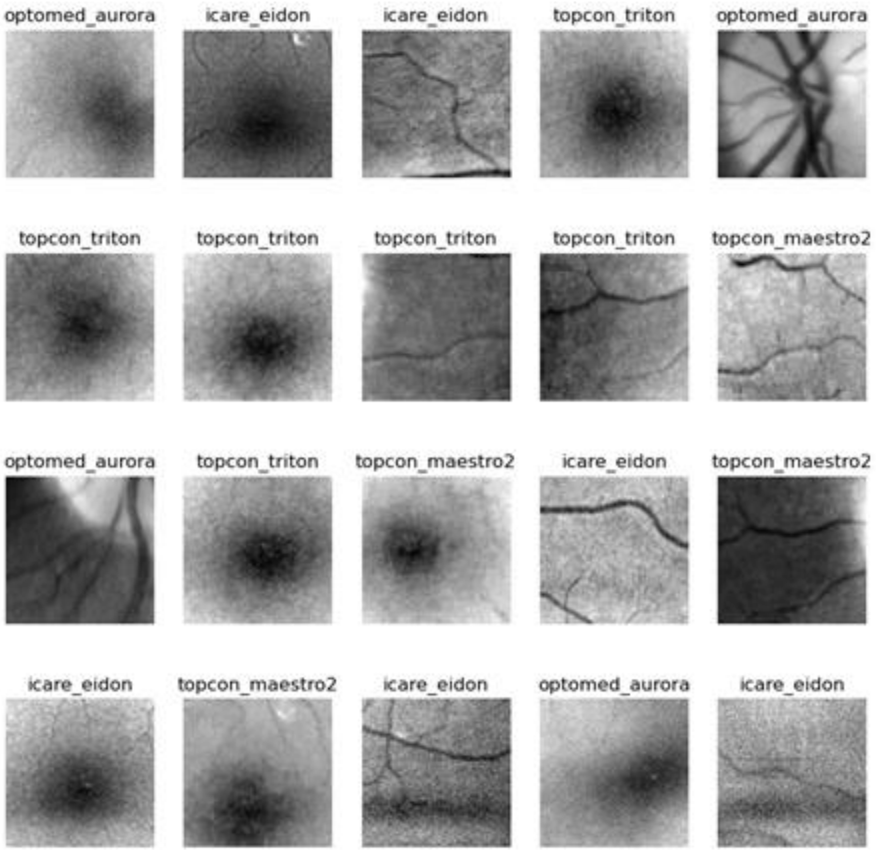
In the AI-READI dataset, we have CFPs from 4 imaging devices

These 4 devices create images that are fairly distinct in color and the classification problem is not very difficult. So we design a more difficult problem.

Can you classify these images derived from the images on the previous screen?






optomed_aurora
icare_eidon
icare_eidon
topcon_triton
optomed_aurora
topcon_triton
topcon_triton
topcon_triton
topcon_triton
topcon_maestro2
optomed_aurora
topcon_triton
topcon_maestro2
icare_eidon
topcon_maestro2
icare_eidon
topcon_maestro2
icare_eidon
optomed_aurora
icare_eidon

So our challenge today is to develop a deep learning model to classify the device from these cropped, grayscale images.

This will demonstrate the power of deep learning.

Today’s plan

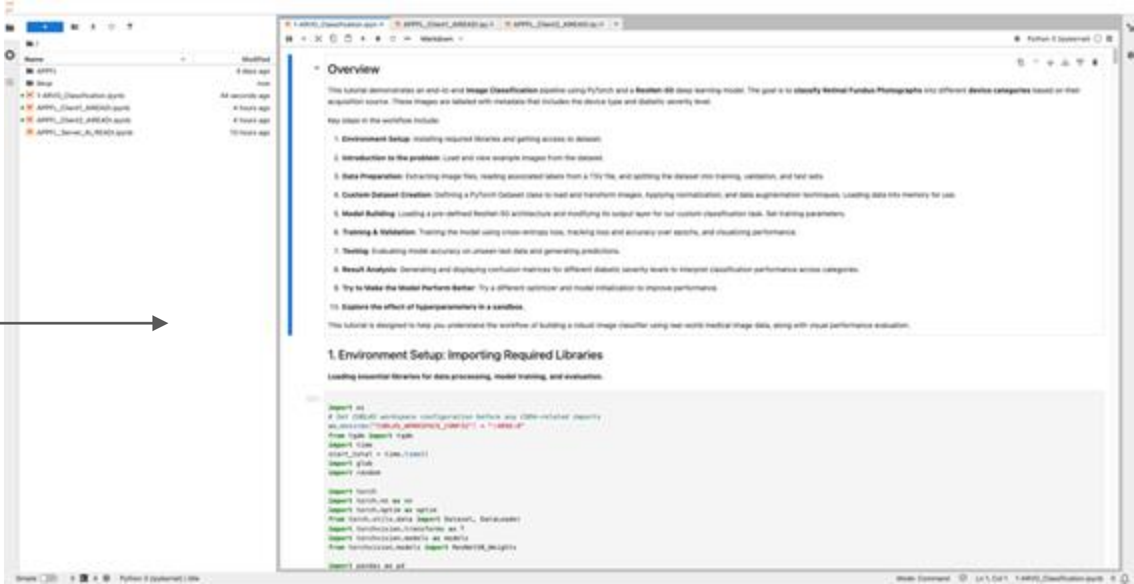
1. Introduce you to Jupyter notebooks
2. Make the plots I just showed you
3. Set up a deep learning model to perform the task we tried to do by eye
4. Train the model
5. Examine the results
6. Explore other settings to improve performance

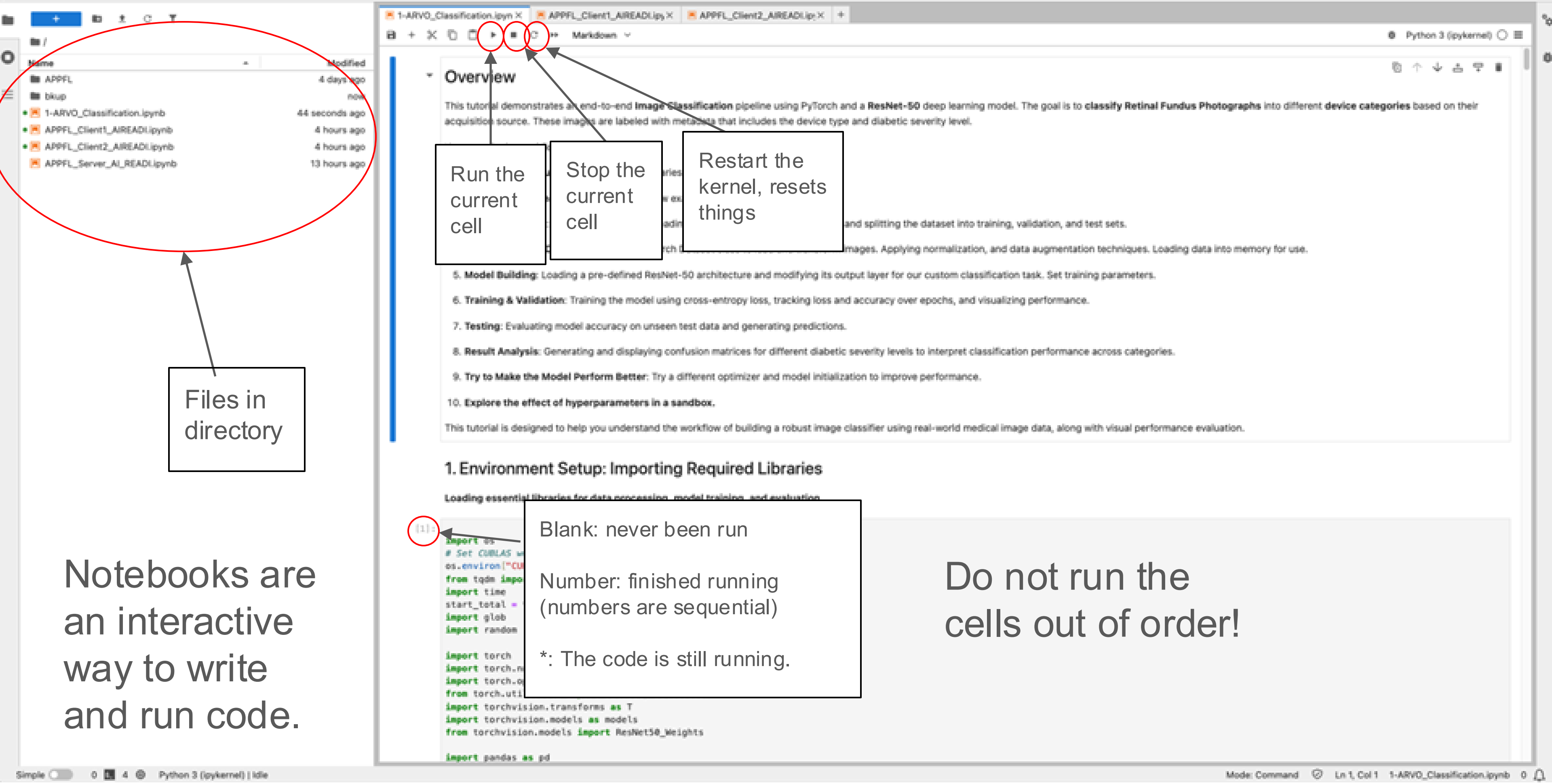
Let’s get everyone logged onto an instance

1. Type the url you were given into an internet browser, example http://5.936.198:8027

1. Type in the password: AIREADI#2025

1. You should be in and see a screen like this





Run the current cell

Stop the current cell

Restart the kernel, resets things

Files in directory

Blank: never been run

Number: finished running (numbers are sequential)

*: The code is still running.

Notebooks are an interactive way to write and run code.

Do not run the cells out of order!

Ok now we are ready to start training a model!

1. Environment Setup: Importing Required Libraries

Loading essential libraries for data processing, model training, and evaluation.

[1]:
```
import os
# Set CUBLAS workspace configuration before any CUDA-related imports
os.environ["CUBLAS_WORKSPACE_CONFIG"] = ":4096:8"
from tqdm import tqdm
import time
start_total = time.time()
import glob
import random

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as T
import torchvision.models as models
from torchvision.models import ResNet50_Weights

import pandas as pd

from PIL import Image

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

from itertools import cycle
from IPython.display import clear_output

torch.use_deterministic_algorithms(True)
```

2. Introduction to the problem: Load and view example images from the dataset.

2a) Problem: Classify device from cropped, grayscale CFP images

We have CFP images from 4 devices: iCare Eidon, Topcon Maestro2, Topcon Triton, and Optomed Aurora.

2b) We can load an image for one participant/eye and view.

[2]:
```
img_eidon = np.asarray(Image.open('APPFL/examples/full/icare_eidon/1270_eidon_uwf_central_cfp_r_1.2.826.0.1.3680043.8.641.1.20240510.232115.43583.jpg'))
img_maestro = np.asarray(Image.open('APPFL/examples/full/topcon_maestro2/1270_maestro2_3d_macula_cfp_r_2.16.840.1.114517.10.5.1.4.907063120240510160806.2.1.jpg'))
img_triton = np.asarray(Image.open('APPFL/examples/full/topcon_triton/1274_triton_macula_12x12_cfp_r_2.16.840.1.114517.10.5.1.4.94005520240514143050.2.1.jpg'))
img_optomed = np.asarray(Image.open('APPFL/examples/full/optomed_aurora/1270_optomed_mac_or_disk_centered_cfp_r_2.25.2183161925995491838121778870991254610818.jpg'))
```

[3]:
```
plt.figure(figsize=(20,50))
plt.subplot(1,4,1)
plt.imshow(img_eidon)
plt.title('iCare Eidon')
plt.axis('off')

plt.subplot(1,4,2)
plt.imshow(img_maestro)
plt.title('Topcon Maestro2')
plt.axis('off')

plt.subplot(1,4,3)
plt.imshow(img_triton)
plt.title('Topcon Triton')
plt.axis('off')

plt.subplot(1,4,4)
plt.imshow(img_optomed)
plt.title('Optomed Aurora')
plt.axis('off')
```

3. Data Preparation ¶

3a) Loading Dataset Labels: We read the labels from a TSV file, which contains metadata about images.

[13]:
```
# importing data label
tsv_path = "APPFL/examples/cfp_images/labels.tsv"
df = pd.read_csv(tsv_path, sep='\t')

# printing top 5
df.head(5)
```

[13]:

	subject_id	device	protocol	laterality	partition	dm_severity	file_path
0	7079	optomed_aurora	mac_or_disk_centered	r	train	pre_diabetes_lifestyle_controlled	optomed_aurora/7079_optomed_mac_or_disk_center...
1	7174	optomed_aurora	mac_or_disk_centered	l	train	oral_medication_and_or_non_insulin_injectable_...	optomed_aurora/7174_optomed_mac_or_disk_center...
2	7103	optomed_aurora	mac_or_disk_centered	r	train	oral_medication_and_or_non_insulin_injectable_...	optomed_aurora/7103_optomed_mac_or_disk_center...
3	7103	optomed_aurora	mac_or_disk_centered	r	train	oral_medication_and_or_non_insulin_injectable_...	optomed_aurora/7103_optomed_mac_or_disk_center...
4	7097	optomed_aurora	mac_or_disk_centered	l	val	oral_medication_and_or_non_insulin_injectable_...	optomed_aurora/7097_optomed_mac_or_disk_center...

[14]:
```
#print the first 20 lines
print(df[['subject_id','device','partition']].head(20))
```

```
    subject_id          device partition
0         7079  optomed_aurora     train
1         7174  optomed_aurora     train
2         7103  optomed_aurora     train
3         7103  optomed_aurora     train
4         7097  optomed_aurora       val
5         4028  optomed_aurora     train
6         4152  optomed_aurora     train
7         4152  optomed_aurora     train
8         1035  optomed_aurora      test
9         1132  optomed_aurora     train
10        7190  optomed_aurora       val
11        7073  optomed_aurora     train
12        1231  optomed_aurora      test
13        1231  optomed_aurora      test
14        1231  optomed_aurora      test
15        7370  optomed_aurora     train
16        7293  optomed_aurora     train
17        1336  optomed_aurora     train
18        1336  optomed_aurora     train
19        1193  optomed_aurora     train
```

3b) Splitting Dataset: The dataset is split into training, validation, and test sets.

```
[15]: # dividing data into diffrent dataframes based on train, validation and test
train_df = df[df["partition"] == "train"].copy()
val_df   = df[df["partition"] == "val"].copy()
test_df  = df[df["partition"] == "test"].copy()

print("Number of data sample \nTraining", len(train_df), "\nValidation", len(val_df), "\nTest", len(test_df))
```

```
Number of data sample
Training 1544
Validation 652
Test 666
```

```
[16]: # Finding unique classes and giving it a number index
# in this case we have 4 classes
unique_classes = sorted(train_df["device"].unique())

# Mapping classes to a particular index
class_to_idx = {cls_name: idx for idx, cls_name in enumerate(unique_classes)}
print("Class to index mapping:", class_to_idx)

train_df["label_idx"] = train_df["device"].map(class_to_idx)
val_df["label_idx"]   = val_df["device"].map(class_to_idx)
test_df["label_idx"]  = test_df["device"].map(class_to_idx)

num_classes = len(unique_classes)
```

```
Class to index mapping: {'icare_eidon': 0, 'optomed_aurora': 1, 'topcon_maestro2': 2, 'topcon_triton': 3}
```

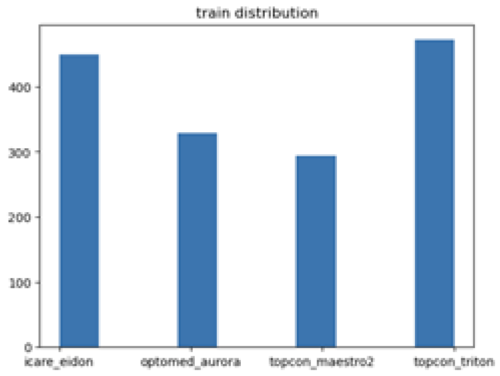
2,862 images total
Roughly 60-20-20 split

54% is from training set -> will be used to train our model
23% is for validation -> will be used to evaluate model during development period
23% is for testing -> will be saved for the end when we evaluate our model

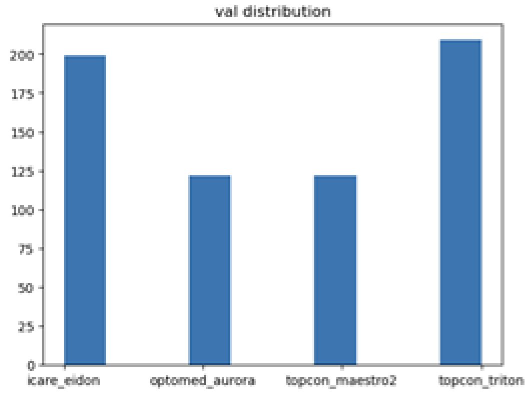
Patient-level split - no leakage between the partitions

3c) Distribution of the Labels:We can plot the distribution of the labels across the three partitions.

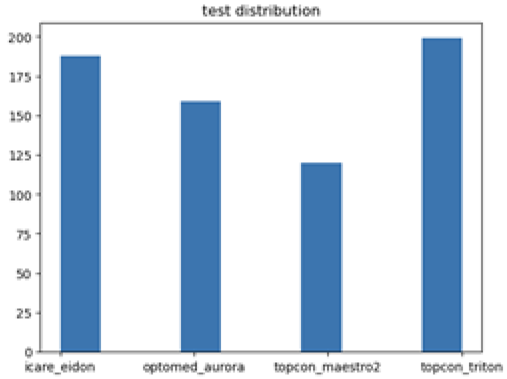
[17]:
```
train_labels = sorted(train_df["device"])
_ = plt.hist(train_labels)
plt.title('train distribution')
plt.show()
```



[18]:
```
val_labels = sorted(val_df["device"])
_ = plt.hist(val_labels)
plt.title('val distribution')
plt.show()
```



[19]:
```
test_labels = sorted(test_df["device"])
_ = plt.hist(test_labels)
plt.title('test distribution')
plt.show()
```



4. Defining the Custom Dataset Class

4a) This class is used for loading images (cropping and converting to grayscale) and applying transformations.

```
*[20]: # Defining a dataloader class, which returns image and its label
class AIREADIDataset_grayscalecrop(Dataset):
    def __init__(self, df, transform=None, preload=False):
        """
        Args:
          df: a DataFrame with at least ['file_path', 'label_idx'] columns
          transform: torchvision transforms (augmentations) to apply
        """
        self.df = df.reset_index(drop=True)
        self.transform = transform
        self.preload = []
        if preload:
            partition = self.df["partition"][0]
            npsavfn = f"APPFL/examples/cfp_images/graycrop-{partition}.npy"
            if os.path.isfile(npsavfn):
                self.preload = np.load(npsavfn)
                print('loaded: ' + partition)
            else:
                for i in tqdm(range(0,len(self.df))):
                    row = self.df.iloc[i]
                    img_path = "APPFL/examples/cfp_images/" + row["file_path"]
                    image = Image.open(img_path).convert("RGB")
                    img_array = np.array(image)
                    #take only the 224x224 center square of the image
                    h, w, _ = img_array.shape
                    top = (h - 224) // 2
                    left = (w - 224) // 2
                    center_crop = img_array[top:top+224, left:left+224]
                    #convert to grayscale
                    gray = np.dot(center_crop[...,:3], [0.2989, 0.5870, 0.1140])
                    gray_3ch = np.stack([gray]*3, axis=-1).astype(np.uint8)
```

4b) Transformation of all images, normalization to ImageNet statistics Augmenting training images to have horizontal flips and rotations for online augementation.

```
[21]: # Defining transformation of image
# Augmenting training data to have horizontal flips and rotations
train_transform = T.Compose([
    T.RandomHorizontalFlip(p=0.5),  # Randomly flip images horizontally
    T.RandomRotation(degrees=15),  # Randomly rotate images
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize using ImageNet stats
])

val_transform = T.Compose([
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Image augmentation, helps prevent overfitting and helps with generalization (especially with small datasets)

4c) Call the loader class with partition filepaths and transforms

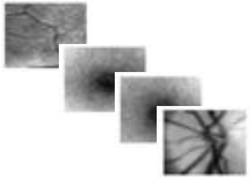
```
[22]: # Loading datasets into memory for faster training
train_dataset = AIREADIDataset_grayscalecrop(train_df, train_transform, preload=True)
val_dataset   = AIREADIDataset_grayscalecrop(val_df, val_transform, preload=True)
test_dataset  = AIREADIDataset_grayscalecrop(test_df, val_transform, preload=True)

loaded: train
loaded: val
loaded: test
```

Quick review of deep learning fundamentals

Loss function:
Cross Entropy

True Labels:
0 = iCare Eidon
1 = Optomed Aurora.
2 = Topcon Maestro2
3= Topcon Triton

ResNet-50

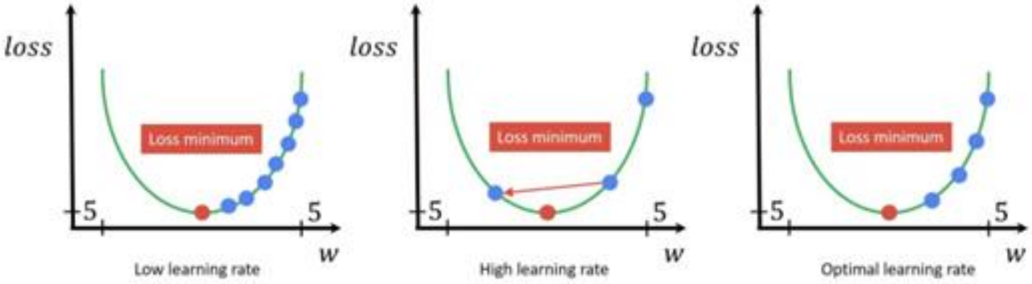


Predicted Labels:
0 = iCare Eidon
1 = Optomed Aurora.
2 = Topcon Maestro2
3 = Topcon Triton

Batch Size:
The number of images to use to compute each step

Number of epochs:
The number time to go through the whole training set

Optimizer and learning rate



5. Model Building

5a) Set batch size and create dataloader instances.

```
[23]: # Define batch size for data loading, batch sie is the number of training examples used during one iteration of model training
batch_size = 64
torch.set_num_threads(os.cpu_count())
num_workers = os.cpu_count()

# Create DataLoader instances for efficient batch processing
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
val_loader   = DataLoader(val_dataset,   batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
test_loader  = DataLoader(test_dataset,  batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
```

5b) Define device to use for training and inference.

```
[24]: # Define the device (GPU if available, else CPU), if this says CPU please raise your hand
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cuda
```

5c) Define methods used during model training, defines what to do each train step, validation inference, and plotting the learning curves.

```
[25]: # Methods to help with model training
def train_step(model, dataloader, optimizer, criterion, device, batch_losses):
    """Performs a single training step over the dataset."""
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    progress_bar = tqdm(dataloader, desc="Training", leave=True)

    for batch_idx, (images, labels, _) in enumerate(progress_bar):
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)

        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

        batch_losses.append(loss.item())
        progress_bar.set_postfix(batch_loss=loss.item())

    epoch_loss = running_loss / total
    epoch_acc = correct / total

    return epoch_loss, epoch_acc

def validate_step(model, dataloader, criterion, device):
    """Performs a single validation step over the dataset."""
    model.eval()
    total_loss = 0.0
    correct = 0
```

5d) Defining model architecture: a pre-existing RESNET50 model architecture.

[26]:
```
#this will make the initializtion the same each time it is run for the same setting of parameters, just run once
def set_seed(seed=42):
    random.seed(seed)                          # Python built-in random
    np.random.seed(seed)                       # NumPy
    torch.manual_seed(seed)                    # PyTorch CPU
    torch.cuda.manual_seed(seed)              # PyTorch CUDA (if using GPU)
    torch.cuda.manual_seed_all(seed)           # All CUDA devices (if using multi-GPU)

set_seed(12)
#Ensure deterministic behavior
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

[27]:
```
# Load the ResNet50 model
model = models.resnet50(weights=None)

# Modify the last layer for our classification task
# Changing to 4 classes as we have 4 devices to classify it into
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Transfer model to device (GPU or CPU)
model = model.to(device)
```

5e) Choose a cost function, an optimizer and set learning rate.

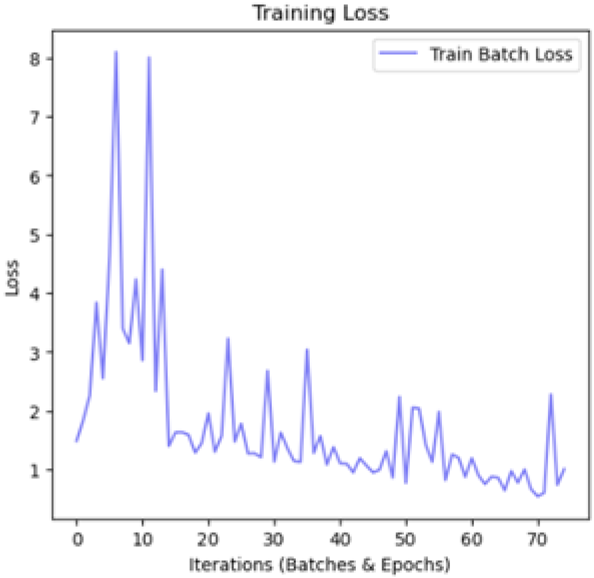
[28]:
```
# Define loss function, the function that we want to minimize to find the best assignment of labels
criterion = nn.CrossEntropyLoss()  #CE quantifies how well a model's predicted probability distribution aligns with the true, or actual, probability distribution of the data

#set the optimier and learning rate (lr), there are different mathemactical methods to minimize loss functions, the learning rate dictates how big a step you take toward the minima
optimizer = optim.SGD(model.parameters(), lr = 1e-2, momentum=0.9)
```

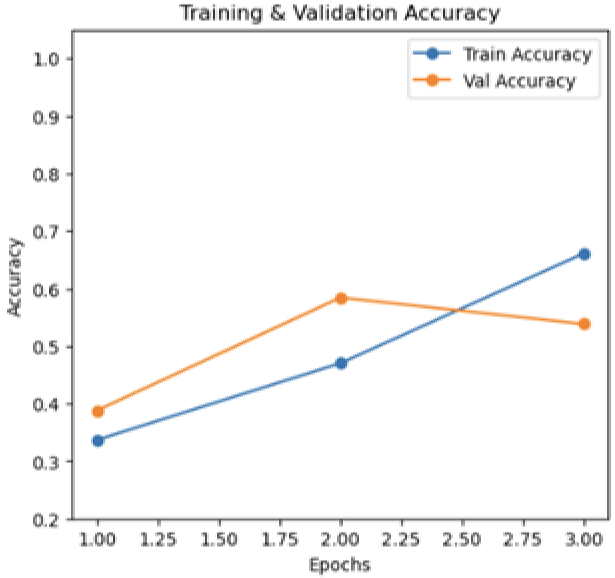
6. Running training: Call training function with desired number of epochs.

```
[29]: #The number of eopch sets how many the model learns over the entire dataset in
start = time.time()
train_model(model=model, train_loader=train_loader,
            val_loader=val_loader,
            optimizer=optimizer,
            criterion=criterion,
            device=device,
            num_epochs=24)
print("Training time: ", time.time() - start)
```

Start training

```
Epoch [3/24] - Train Loss: 1.0995, Train Acc: 0.6613 | Val Loss: 4.3258, Val Acc: 0.5383
```





7. Testing model: Perform inference with model on validation dataset

7a) Our model hasn't seen this data during training so gives us an indication on how it will perform on new data

[30]:

```
# -------------------- VAL --------------------
all_preds = []
all_labels = []
all_dm_severity = []
print("Evaluating on val set...")
if len(val_df) > 0:
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for images, labels, dm_severity in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * images.size(0)

            _, preds = torch.max(outputs, 1)
            val_correct += (preds == labels).sum().item()
            val_total += labels.size(0)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            all_dm_severity.extend(dm_severity)

    val_loss /= val_total
    val_acc = val_correct / val_total
    print(f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")
```

```
Evaluating on val set...
Val Loss: 1.3090, Val Acc: 0.7914
```

7b) Saving test set results to a dataframe for use below.

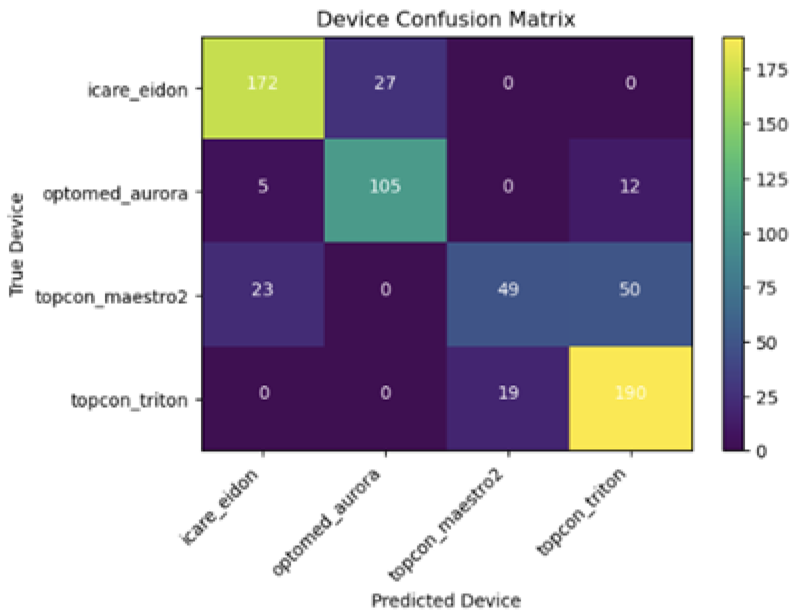
[31]:

```
# Create a DataFrame to store results
df_results_val1 = pd.DataFrame({
    'true_device': all_labels,
    'pred_device': all_preds,
    'dm_severity': all_dm_severity
})
```

8a) Plot confusion matrix of device type for the val set. This allows for interrogation of model performance on each class (device).

```
[32]: #define a function to plot a confusion matrix (cm)
def plot_confusion_matrix(cm, device_names, title="Confusion Matrix"):
    plt.figure()
    plt.imshow(cm, interpolation='nearest', aspect='auto')
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(device_names))
    plt.xticks(tick_marks, device_names, rotation=45, ha='right')
    plt.yticks(tick_marks, device_names)

    # Optionally annotate cells with counts
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, str(cm[i, j]),
                     horizontalalignment="center",color="white")
                     #color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.xlabel('Predicted Device')
    plt.ylabel('True Device')
    plt.show()
```

```
[33]: # Generate confusion matrices with confusion_matrix function from scikit-learn
true_devs = df_results_val1['true_device'].values
pred_devs = df_results_val1['pred_device'].values

cm = confusion_matrix(true_devs, pred_devs)

idx_to_device = {v: k for k, v in class_to_idx.items()}
device_names = [idx_to_device[i] for i in sorted(idx_to_device.keys())]

#calls function witten above
plot_confusion_matrix(cm, device_names, title=f"Device Confusion Matrix")
```

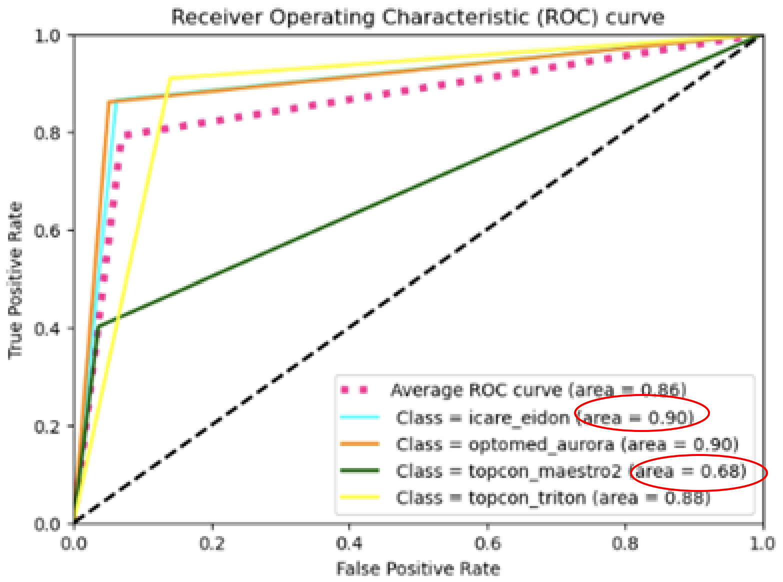


8b) Plot Receiver-Operator Curve (ROC)

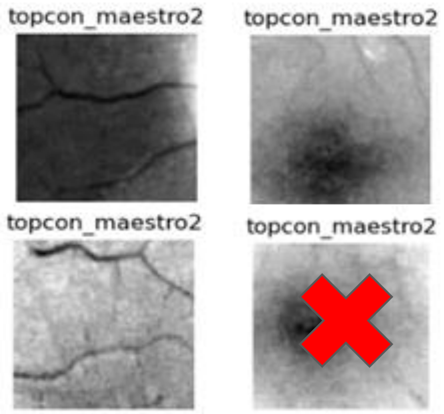
An ROC curve plots the rate of false positives versus the rate of true positives. In our case, we just have one (FPR,TPR) pair and the (0,0) - > everything is predicted 0 and (1,1) points -> everything predicted 1.

[34]:
```
def plot_roc_curve(y_test, y_pred, class_labels):

  n_classes = len(np.unique(y_test))
  y_test = label_binarize(y_test, classes=np.arange(n_classes))
  y_pred = label_binarize(y_pred, classes=np.arange(n_classes))

  # Compute ROC curve and ROC area for each class
  fpr = dict()
  tpr = dict()
  roc_auc = dict()
  for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

[35]:
```
plot_roc_curve(df_results_val1['true_device'].values,df_results_val1['pred_device'].values, device_names)
```



If we consider the topcon_maestro2 class:
True positive - correctly labeled topcon_maestro2
False positive - image incorrectly labeled topcon_maestro2



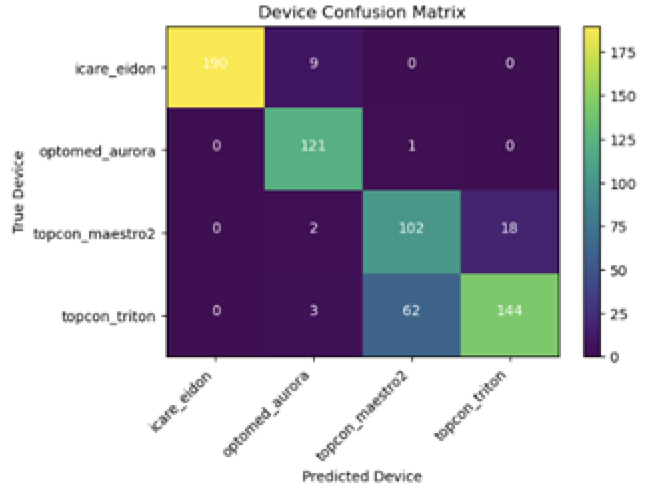
9. Improve model performace

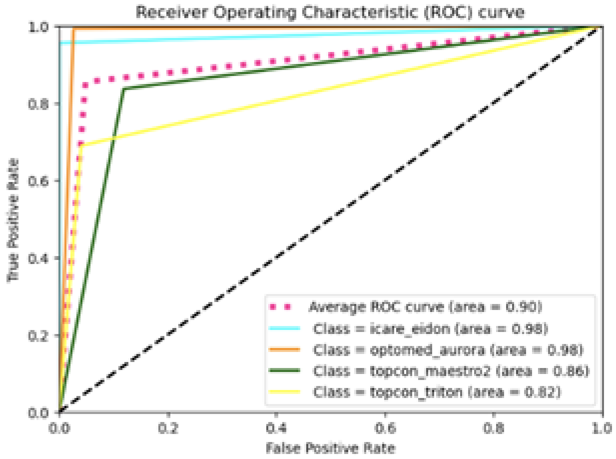
9a) Let's explore a different optimizer and learning rate.

Start training

```
[36]: # Define batch size for data loading
batch_size = 64
torch.set_num_threads(os.cpu_count())
num_workers = os.cpu_count()

# Create DataLoader instances for efficient batch processing
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
val_loader   = DataLoader(val_dataset,   batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
test_loader  = DataLoader(test_dataset,  batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))

# Define the device (GPU if available, else CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Load the ResNet50 model
model = models.resnet50(weights=None)
# Modify the last layer for our classification task
# Changing to 4 classes as we have 4 devices to classify it into
model.fc = nn.Linear(model.fc.in_features, num_classes)
# Transfer model to device (GPU or CPU)
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()

#set the optimizer and learning rate (lr)
optimizer = optim.Adam(model.parameters(), lr=7e-6, weight_decay=1e-3)   #<- this is what we are changing!!

start = time.time()
train_model(model=model, train_loader=train_loader,
            val_loader=val_loader,
            optimizer=optimizer,
            criterion=criterion,
            device=device,
            num_epochs=24)
print("Training time: ", time.time() - start)
```

Run the next 3 cells



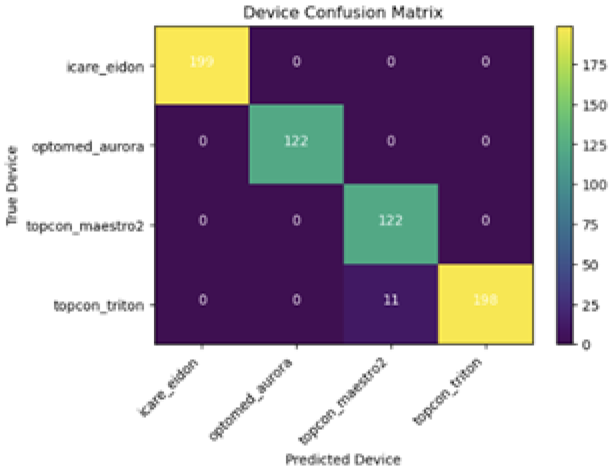


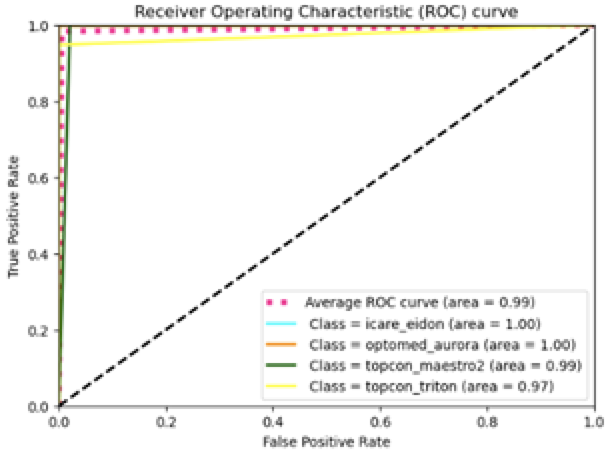
Run the next 3 cells

9b) Second the weight initialisation will be changed by using the pretrained weights (ImageNet) for the ResNet50.

Start training

```
# Define batch size for data loading
batch_size = 64
torch.set_num_threads(os.cpu_count())
num_workers = os.cpu_count()

# Create DataLoader instances for efficient batch processing
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
val_loader   = DataLoader(val_dataset,   batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
test_loader  = DataLoader(test_dataset,  batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))

# Define the device (GPU if available, else CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Load the ResNet50 model
model = models.resnet50(weights = ResNet50_Weights.IMAGENET1K_V2) #<- this is what we are changing!!
# Modify the last layer for our classification task
# Changing to 4 classes as we have 4 devices to classify it into
model.fc = nn.Linear(model.fc.in_features, num_classes)
# Transfer model to device (GPU or CPU)
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()

#set the optimier and learning rate (lr)
optimizer = optim.Adam(model.parameters(), lr=7e-6, weight_decay=1e-3)

start = time.time()
train_model(model=model, train_loader=train_loader,
            val_loader=val_loader,
            optimizer=optimizer,
            criterion=criterion,
            device=device,
            num_epochs=30)
print("Training time: ", time.time() - start)
```



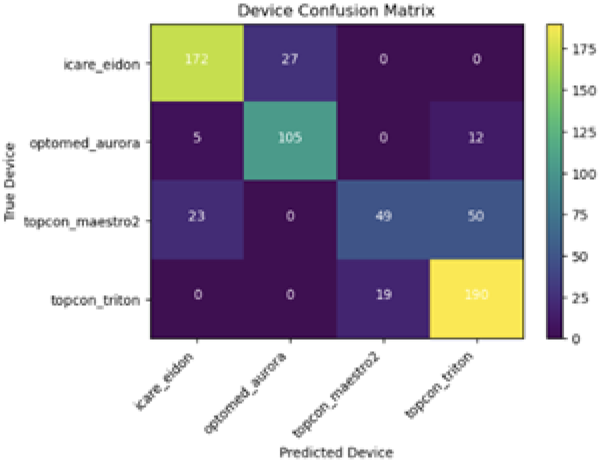


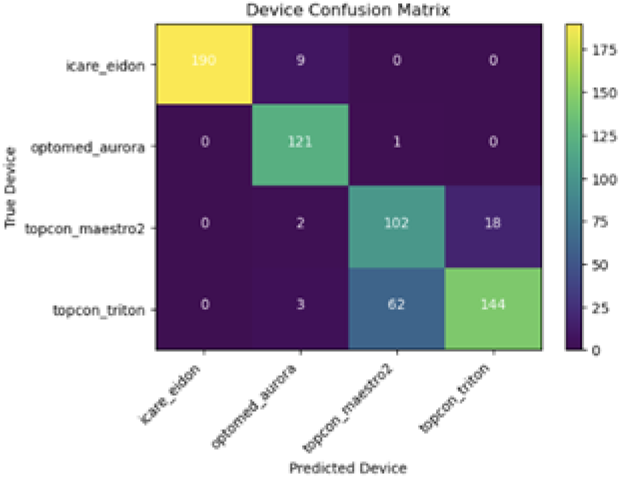
A decision to make!

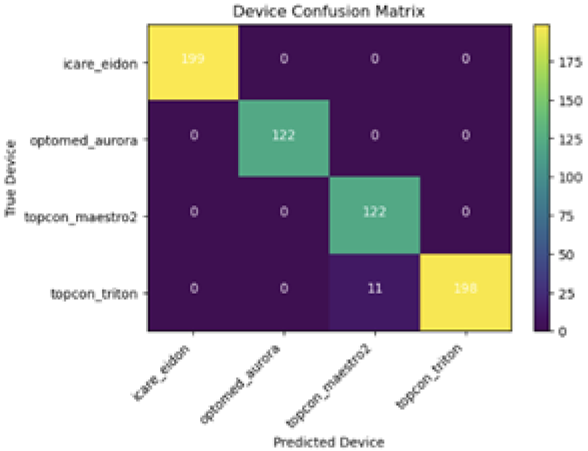
Model 1: Val accuracy: 79.14%

Model 2: Val accuracy: 85.43%

Model 3: Val accuracy: 98.31%





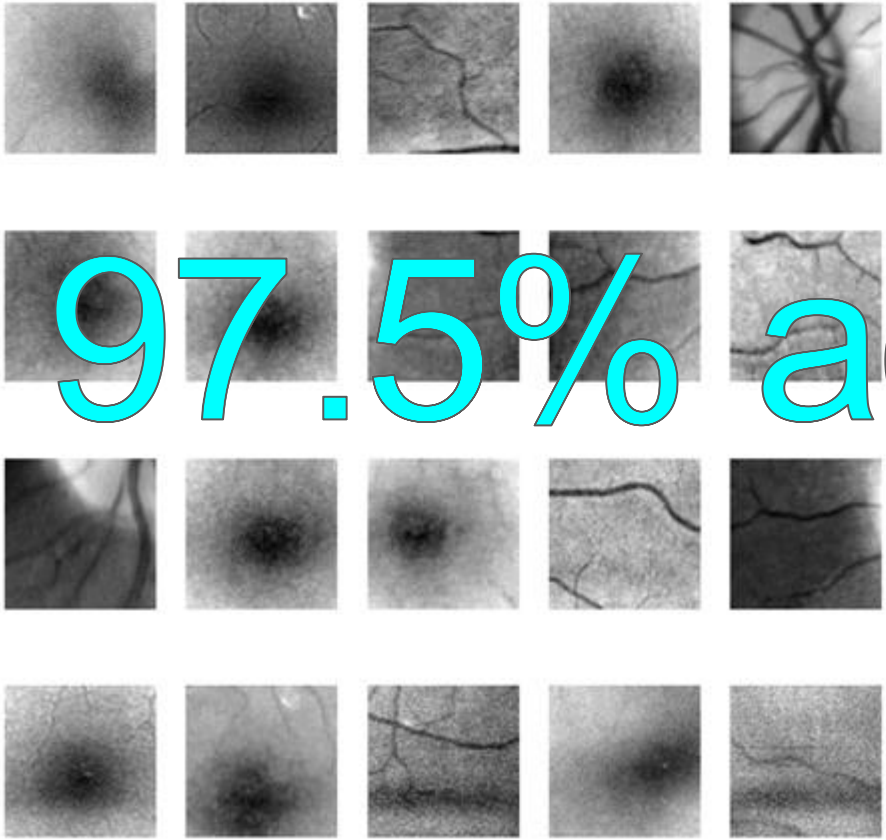


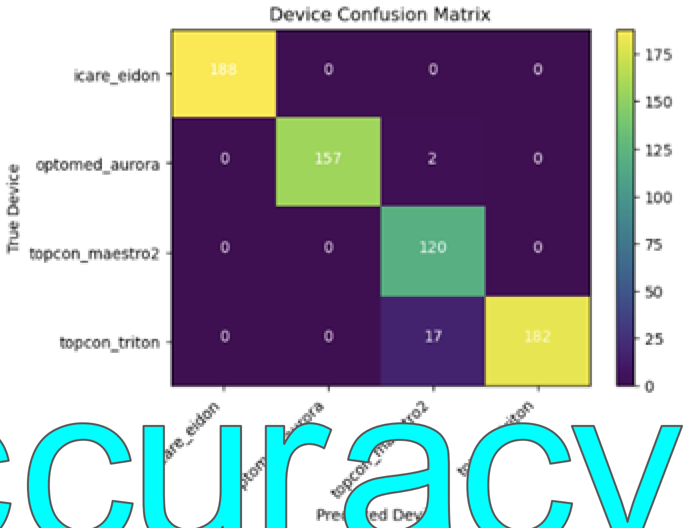
10. Choose a model to evaluate on the test data

The model has not seen these images so this is the least biased way to evaluate a model's perfroamce. But you can only run one model on your test set, so choose carefully.

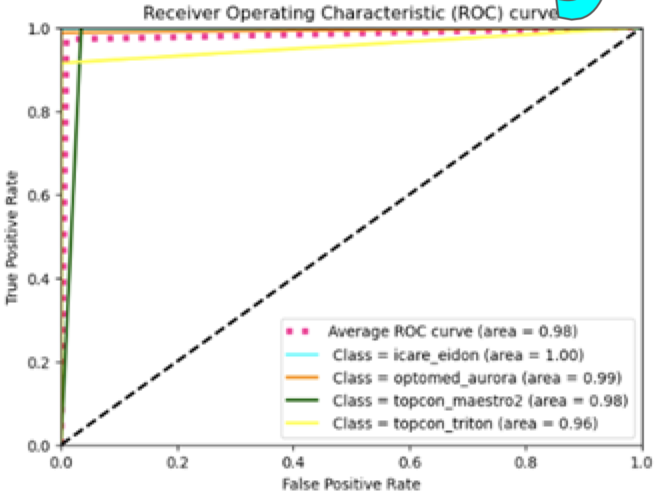
[44]:
```
# Evaluate on the test set

all_preds = []
all_labels = []
all_dm_severity = []
print("Evaluating on test set...")
if len(test_df) > 0:
    model.eval()
    test_loss = 0.0
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for images, labels, dm_severity in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item() * images.size(0)

            _, preds = torch.max(outputs, 1)
            test_correct += (preds == labels).sum().item()
            test_total += labels.size(0)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            all_dm_severity.extend(dm_severity)

    test_loss /= test_total
    test_acc = test_correct / test_total
    print(f"Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.4f}")

    # Create a DataFrame to store results
df_results_test = pd.DataFrame({
    'true_device': all_labels,
    'pred_device': all_preds,
    'dm_severity': all_dm_severity
})
```

[45]:
```
# Generate confusion matrices with confusion_matrix function from scikit-learn
true_devs = df_results_test['true_device'].values
pred_devs = df_results_test['pred_device'].values
cm = confusion_matrix(true_devs, pred_devs)

idx_to_device = {v: k for k, v in class_to_idx.items()}
device_names = [idx_to_device[i] for i in sorted(idx_to_device.keys())]

#calls function witten above
plot_confusion_matrix(cm, device_names, title=f"Device Confusion Matrix")
```





97.5% accuracy!!



If you have time…..

11. Sandbox: Explore hyperparameter settings.

This code was set up for you to explore what happens when you change hyperparameters.

Things you could try:

1. Change data augmentation parameters.
2. Change batch size (warning you may run into memory constraints)
3. Change the initialization seed to any number
4. Change the initialization of the weights (model = models.resnet50(weights = ResNet50_Weights.IMAGENET1K_V2))
5. Change the optimizer and/or learning rate (https://pytorch.org/docs/main/optim.html)
6. Change the number of epocs (you want to see model convergence, training curve is flat at end)

```
# Defining transformation of image
# Augmenting training data to have horizontal flips and rotations
train_transform = T.Compose([
    T.RandomHorizontalFlip(p=0.5),  # Randomly flip images horizontally
    T.RandomRotation(degrees=15),  # Randomly rotate images
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize using ImageNet stats
])
val_transform = T.Compose([
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Loading datasets into memory for faster training
train_dataset = AIREADIDataset_grayscalecrop(train_df, train_transform, preload=True)
val_dataset   = AIREADIDataset_grayscalecrop(val_df, val_transform, preload=True)
test_dataset  = AIREADIDataset_grayscalecrop(test_df, val_transform, preload=True)

# Define batch size for data loading
batch_size = 64
torch.set_num_threads(os.cpu_count())
num_workers = os.cpu_count()

# Create DataLoader instances for efficient batch processing
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
val_loader   = DataLoader(val_dataset,   batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))
test_loader  = DataLoader(test_dataset,  batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True, worker_init_fn=lambda _: np.random.seed(42))

# Define the device (GPU if available, else CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

#this will make the initializtion the same each time it is run for the same setting of parameters

set_seed(12)

# Load the ResNet50 model
model = models.resnet50(weights=None)
# Modify the last layer for our classification task
# Changing to 4 classes as we have 4 devices to classify it into
model.fc = nn.Linear(model.fc.in_features, num_classes)
# Transfer model to device (GPU or CPU)
model = model.to(device)
```

Acknowledgements

Ravi K. Madduri
Argonne National Laboratory

Aditya Sinha
University of Illinois, Urbana-Champaign

Aaron Lee
University of Washington

All the Lee Lab who helped us test!

Go check out the notebooks later! https://github.com/AdioosinUIUC/aireadi-course