RETINAL DISEASE CLASSIFICATION

GROUP 20

Nguyen Quang
Hoai Thuong
Viet Hoang

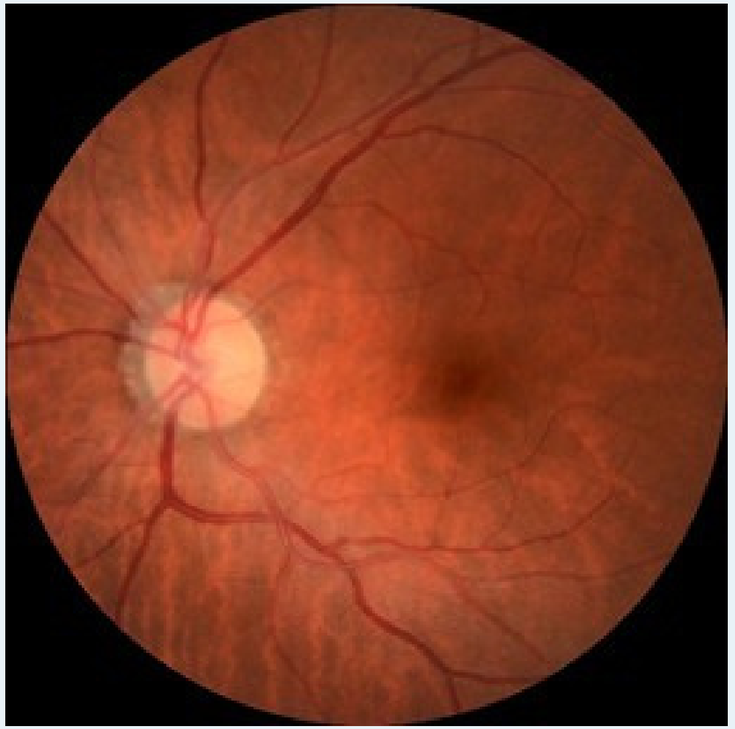
RETINAL DISEASE

=

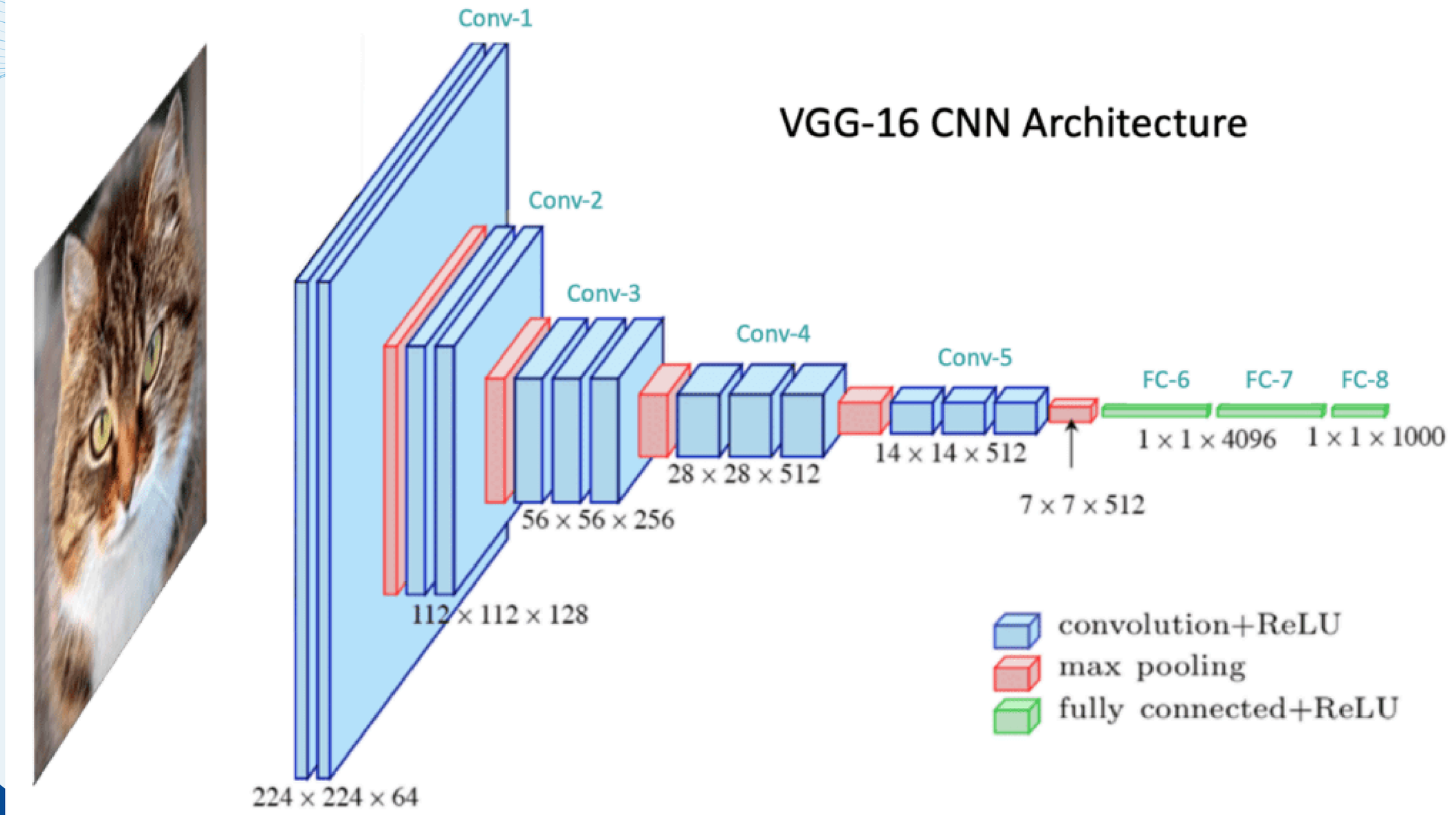
LEADING CAUSES OF PREVENTABLE BLINDNESS GLOBALLY.

Blood Vessels

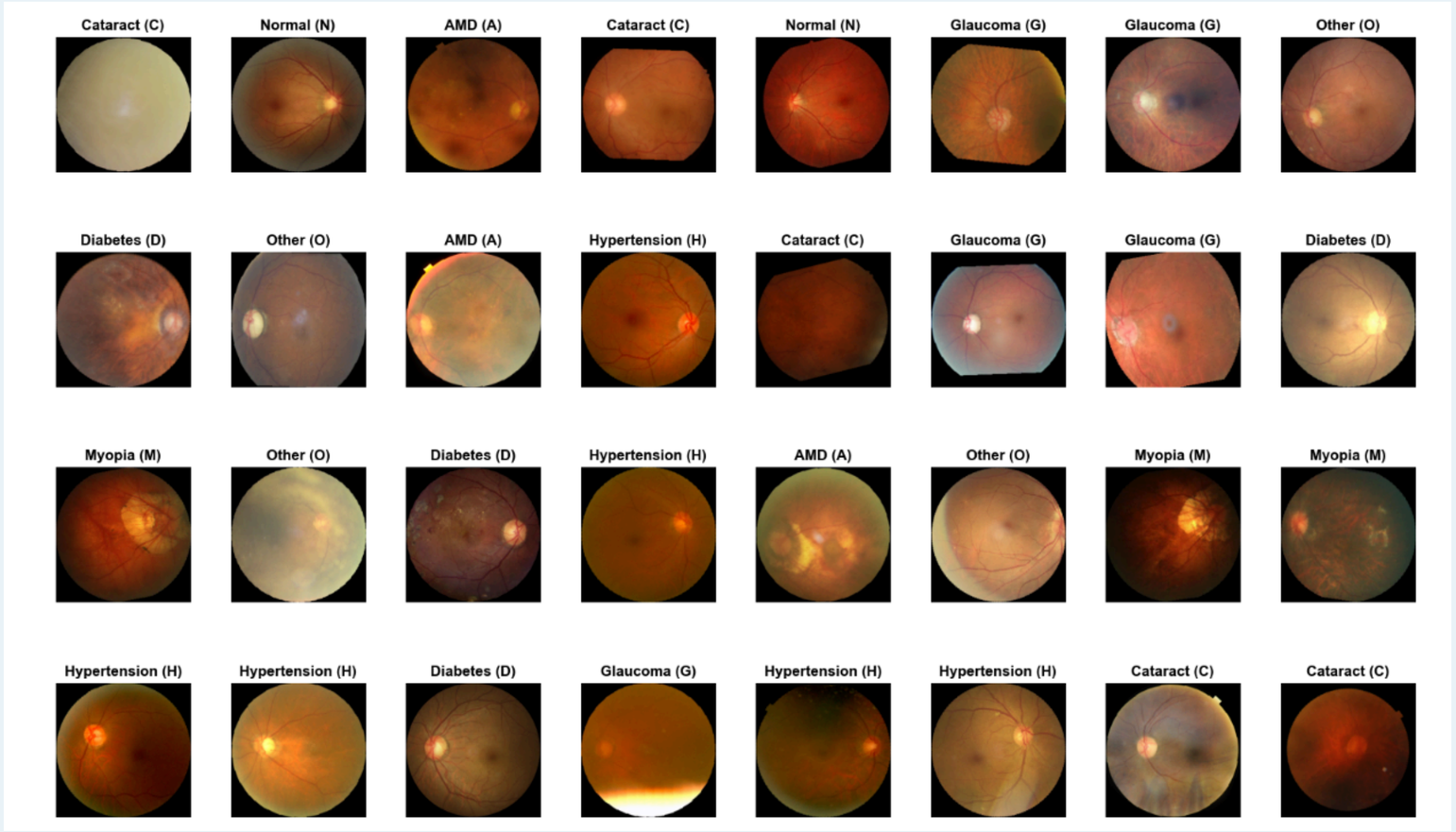
Lesion



Complicated!


VGG-16 CNN Architecture
Conv-1
Conv-2
Conv-3
Conv-4
Conv-5
FC-6
FC-7
FC-8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU

IDENTIFY INPUTS AND OUTPUTS



~10000 images of retinal fundus images

For efficiency:
512 x 512 → 254 x 254

0: 'Normal (N)', 1: 'Diabetes (D)', 2: 'Glaucoma (G)', 3: 'Cataract (C)', 4: 'AMD (A)', 5: 'Hypertension (H)', 6: 'Myopia (M)', 7: 'Other (O)'

IDENTIFY INPUTS AND OUTPUTS

file csv

```
raw_df = pd.read_csv('label_images.csv')
raw_df.head()
```

0.1s Python

	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	label	target	images
0	0.0	69.0	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	0_right.jpg
1	1.0	57.0	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1_right.jpg
2	2.0	42.0	Male	2_left.jpg	2_right.jpg	laser spot，moderate non proliferative retinopathy	moderate non proliferative retinopathy	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	2_right.jpg
3	4.0	53.0	Male	4_left.jpg	4_right.jpg	macular epiretinal membrane	mild nonproliferative retinopathy	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	4_right.jpg
4	5.0	50.0	Female	5_left.jpg	5_right.jpg	moderate non proliferative retinopathy	moderate non proliferative retinopathy	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	5_right.jpg

```
describe_df = raw_df[['ID', 'label', 'images']]
describe_df.describe(include='all')
```

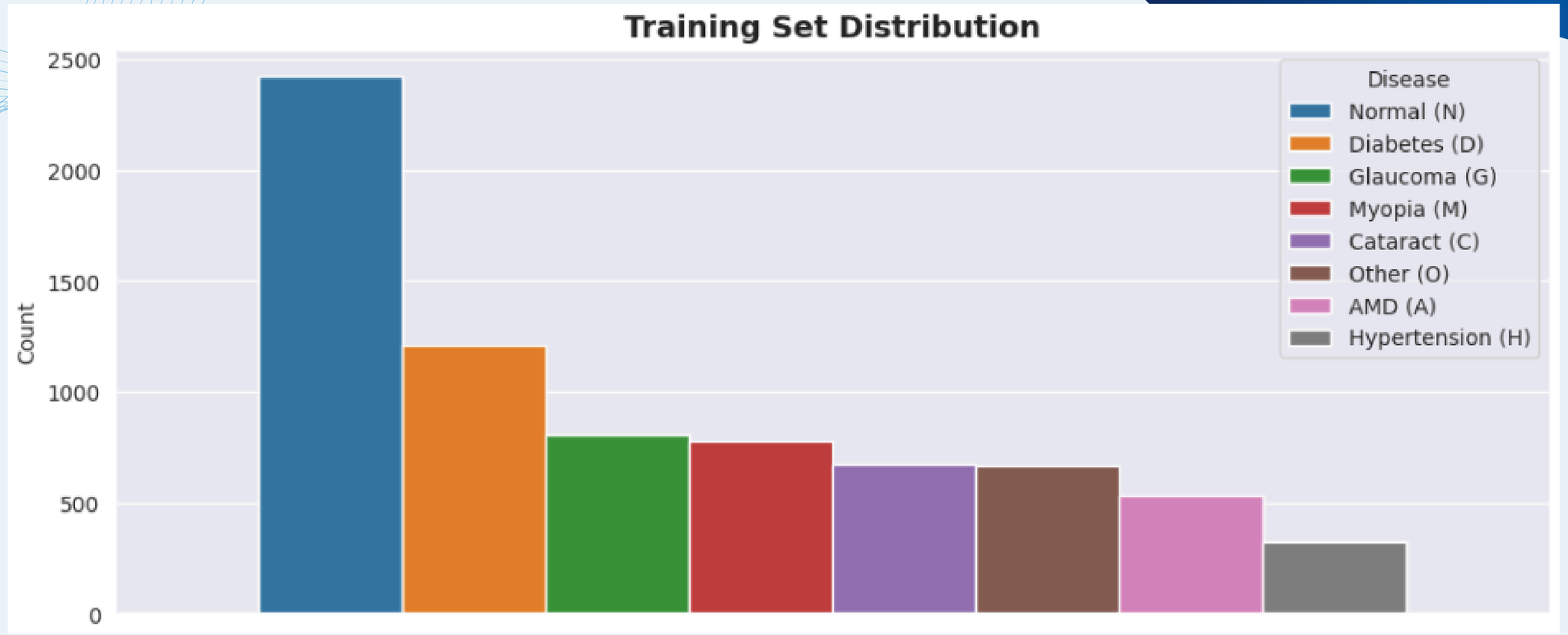
Missing Entries!

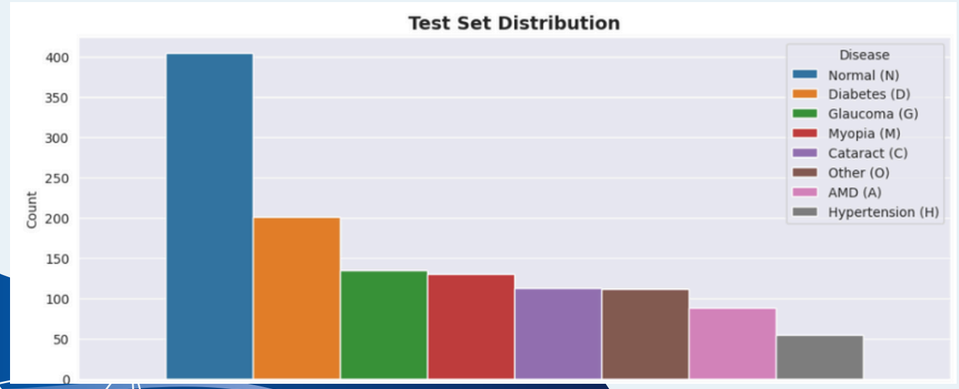
	ID	label	images
count	6392.000000	9868	9868
unique	NaN	8	9868
top	NaN	['N']	NEWOTHER_EXTRA_DATA993.png
freq	NaN	3230	1

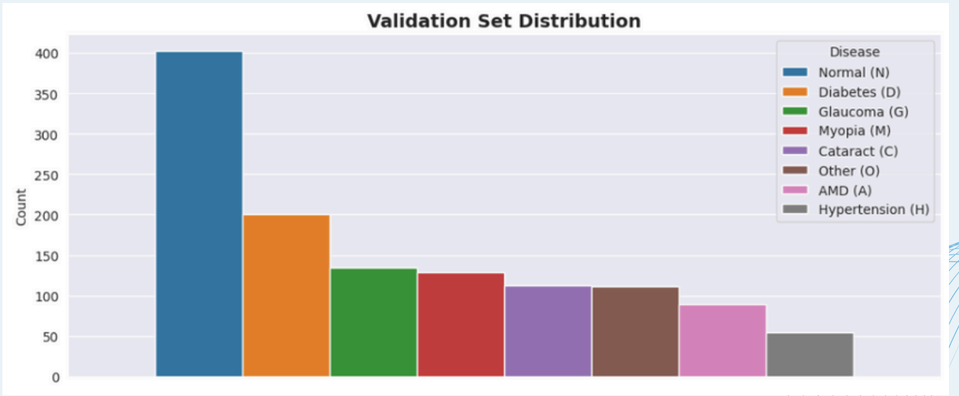
```python
# Train-Test-Val Split
train_df, temp_df = train_test_split(raw_df, test_size=0.25, random_state=42, stratify=raw_df['label'])
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42, stratify=temp_df['label'])

print(f"Train size: {len(train_df)} (Used for training)")
print(f"Val size:   {len(val_df)}   (Used for validating)")
print(f"Test size:  {len(test_df)}  (Used for final score)")
```

```
Train size: 7401 (Used for training)
Val size:   1233   (Used for early stopping)
Test size:  1234  (Used for final score)
```

STRATIFIED SPLITTING: ENSURE FAIR DISTRIBUTION


Training Set Distribution
Count
2500
2000
1500
1000
500
0
Disease
Normal (N)
Diabetes (D)
Glaucoma (G)
Myopia (M)
Cataract (C)
Other (O)
AMD (A)
Hypertension (H)

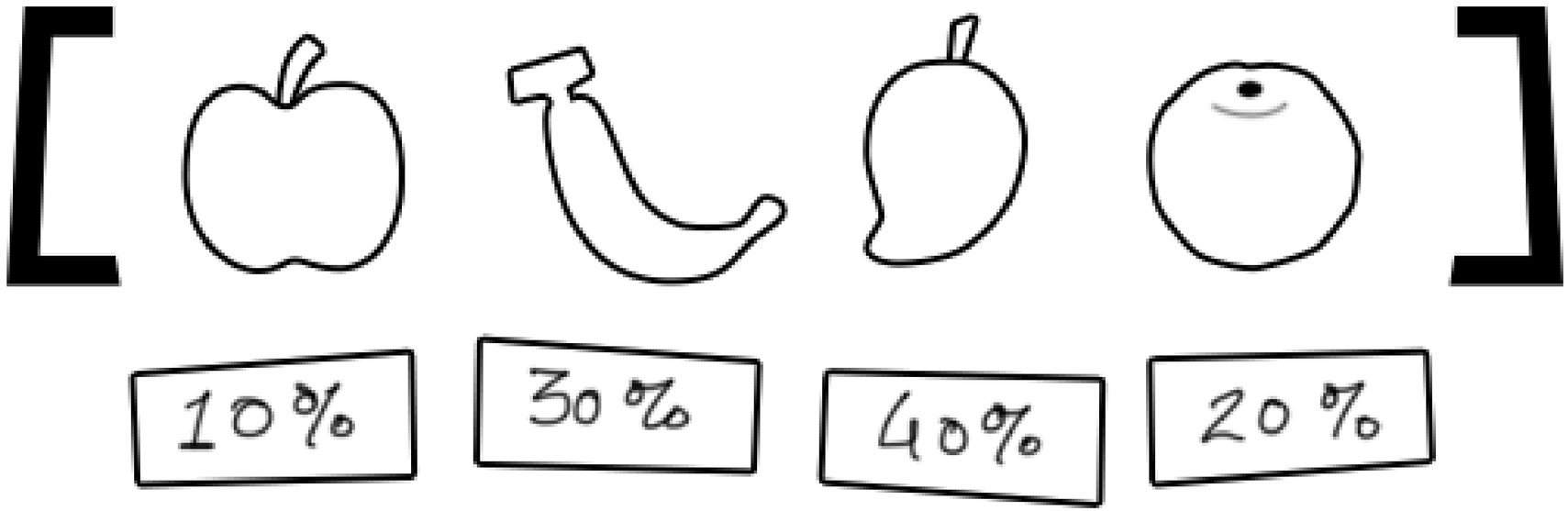

Test Set Distribution
Count
400
350
300
250
200
150
100
50
0
Disease
Normal (N)
Diabetes (D)
Glaucoma (G)
Myopia (M)
Cataract (C)
Other (O)
AMD (A)
Hypertension (H)


Validation Set Distribution
Count
400
350
300
250
200
150
100
50
0
Disease
Normal (N)
Diabetes (D)
Glaucoma (G)
Myopia (M)
Cataract (C)
Other (O)
AMD (A)
Hypertension (H)

OPTIMIZATION TECHNIQUES

WEIGHTED RANDOM SAMPLER (DATA LEVEL)

CLASS-WEIGHTED LOSS FUNCTION (ALGORITHM LEVEL)



RARE

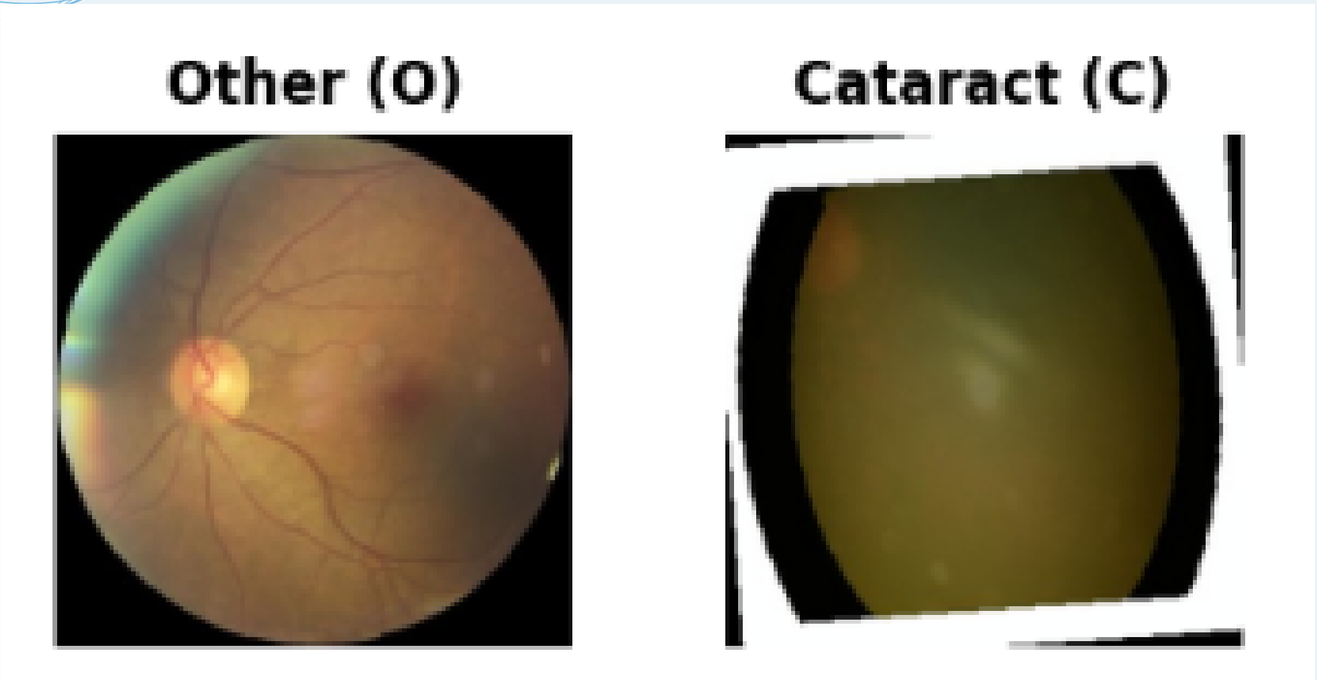
COMMON

HUGE WEIGHT

TINY WEIGHT

WEIGHTED RANDOM SAMPLER RESULT

DATA AUGMENTATION




```
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=10),
    transforms.RandomAdjustSharpness(sharpness_
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

ROTATED

NEURAL NETWORK ARCHITECTURE

```python
def __init__(self, num_class):
    super(ClassificationCnn, self).__init__()
    self.num_class = num_class

    # 4 blocks to detect patterns.
    # Each reduces image size by half, doubles the number of "filters".
    self.conv = nn.Sequential(
        # Block 1
        # Input: (3, 256, 256) -> Output: (16, 128, 128)
        nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 2
        # Input: (16, 128, 128) -> Output: (32, 64, 64)
        nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 3
        # Input: (32, 64, 64) -> Output: (64, 32, 32)
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),

        # Block 4
        # Input: (64, 32, 32) -> Output: (128, 16, 16)
        nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
        nn.BatchNorm2d(128),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
```

NEURAL NETWORK ARCHITECTURE

```
# Takes the features found above and makes a decision.
# Input Size: 128 channels * 6 width * 6 height
self.fc = nn.Sequential(
    # Layer 1: Connection from feature maps to hidden neurons
    nn.Linear(128 * 16 * 16, 256),
    nn.ReLU(),

    # Dropout: Prevents the model from overfitting
    nn.Dropout(0.25),

    # Layer 2: Compress info further
    nn.Linear(256, 128),
    nn.ReLU(),

    # Output: Final decision scores for each class
    nn.Linear(128, self.num_class)
)
```

NEURAL NETWORK ARCHITECTURE

```
# Takes the features found above and makes a decision.
# Input Size: 128 channels * 6 width * 6 height
self.fc = nn.Sequential(
    # Layer 1: Connection from feature maps to hidden neurons
    nn.Linear(128 * 16 * 16, 256),
    nn.ReLU(),

    # Dropout: Prevents the model from overfitting
    nn.Dropout(0.25),

    # Layer 2: Compress info further
    nn.Linear(256, 128),
    nn.ReLU(),

    # Output: Final decision scores for each class
    nn.Linear(128, self.num_class)
)
```

NEURAL NETWORK ARCHITECTURE

```
def forward(self, x):
    """
    Forward pass defines how data moves through the graph.
    """
    # Extract features
    x = self.conv(x)

    # Flatten: Turn (Batch, 128, 6, 6) -> (Batch, 128)
    x = x.view(x.size(0), -1)

    # Classify
    x = self.fc(x)
    return x
```

TRAINING

```
# Criterion: Measures how wrong the model is
criterion = nn.CrossEntropyLoss()

# Optimizer: Adjusts the weights based on the errors.
optimizer = torch.optim.SGD(model.parameters(), lr=1e-2, momentum=0.9)

# Scheduler: Lower learning_rate if no improvement
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=3)
```

TRAINING

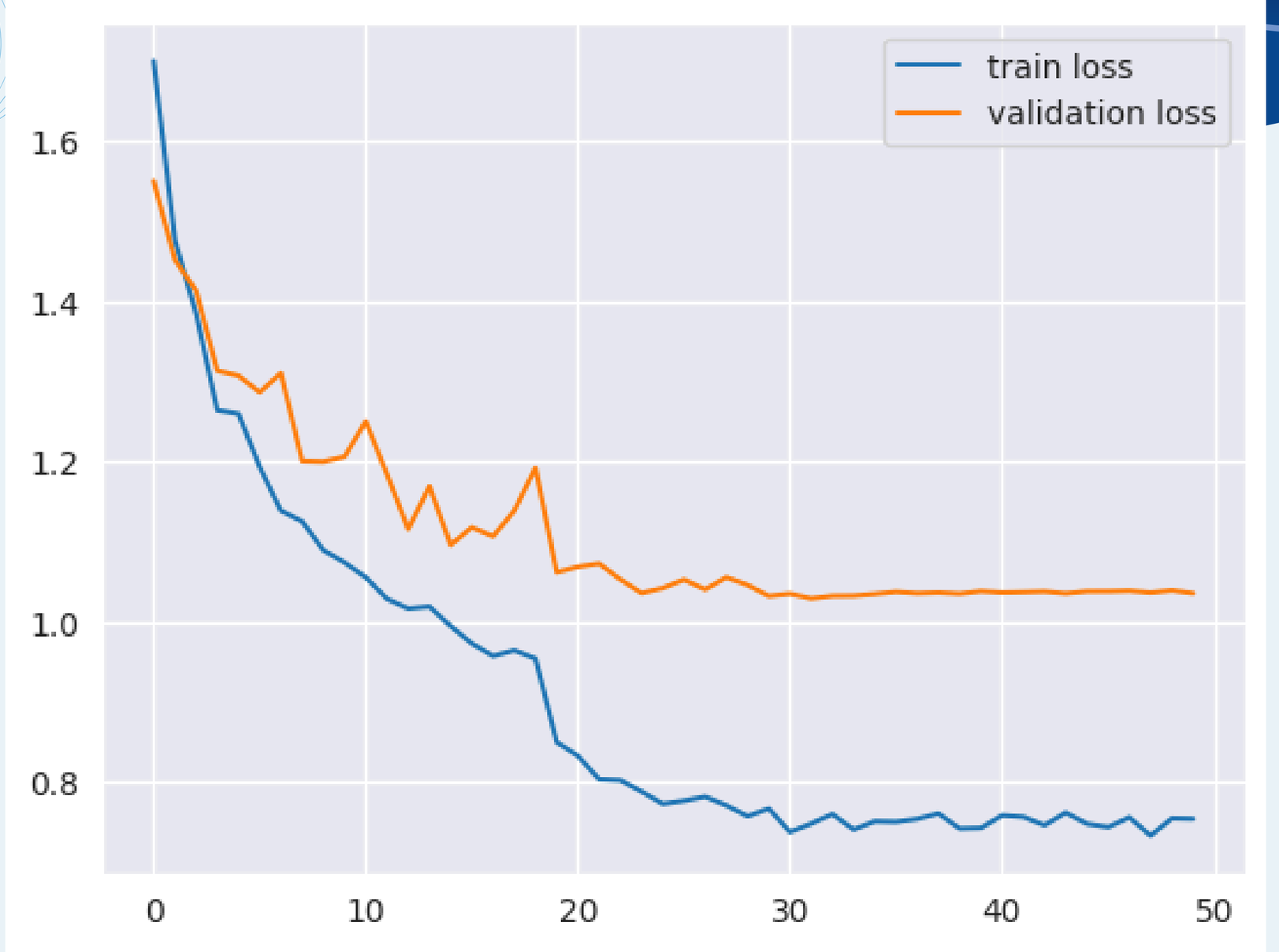
```python
# Training

# Track corrects and loss per epoch to calculate averages
train_corrects = 0
train_loss = 0.0

model.train()

# Wrap with tqdm for progress bar
for inputs, targets in tqdm(train_loader, desc=f'Training Epoch: {epoch + 1}/{N_EPOCHS}'):

    inputs, targets = inputs.to(device), targets.to(device)

    # Reset gradients for each batch
    optimizer.zero_grad()

    # Forward Pass: Ask model for prediction
    outputs = model(inputs)
```

TRAINING

```
# Validation

# Disable gradient calculation & dropout, lock batch norm
with torch.no_grad():
    val_corrects = 0
    val_loss = 0.0
    model.eval()
```

```
val_loss = val_loss / len(val_loader.dataset)
val_acc = val_corrects / len(val_loader.dataset)

scheduler.step(val_loss)

if val_acc > best_val_acc:
    best_val_acc = val_acc

    # Save copy of  performing weights
    best_model_wts = copy.deepcopy(model.state_dict())
```

EVALUATION




```
Train Accuracy: 0.5426, Validation Accuracy: 0.5556, Test Accuracy: 0.5478
```

```
[[281  14  38  43   6   8   5   9]
 [145  17   2  12   1  12   2  10]
 [ 24   0 103   4   0   2   1   0]
 [  2   0   4  88  16   0   2   0]
 [ 18   2   5  10  43   2   7   1]
 [ 11   0   3   1   1  38   0   0]
 [  4   1   8   2  11   1 103   0]
 [ 63   5   1  15  20   2   2   3]]
```

	precision	recall	f1-score	support
0	0.51	0.70	0.59	404
1	0.44	0.08	0.14	201
2	0.63	0.77	0.69	134
3	0.50	0.79	0.61	112
4	0.44	0.49	0.46	88
5	0.58	0.70	0.64	54
6	0.84	0.79	0.82	130
7	0.13	0.03	0.04	111
accuracy			0.55	1234
macro avg	0.51	0.54	0.50	1234
weighted avg	0.51	0.55	0.50	1234

STRENGTH

- **DATA PIPELINE**
- **STRATIFIED SPLITTING**
- **WEIGHTED RANDOM SAMPLING**
- **CUSTOM VGG-LIKE ARCHITECTURE FROM SCRATCH**

WEAKNESSES

- **RESOLUTION CONSTRAINTS**
- **THE 'OTHER' CLASS**

FUTURE WORK

- **TRANSFER LEARNING**
- **HIGHER RESOLUTION**

THANK YOU