CNN Competition for Glaucoma Detection - Submission Instructions

Overview

Welcome to the Glaucoma Detection CNN Competition! This document contains all the information you need to submit your trained model for evaluation.

Submission Structure

Your submission must be a **single folder** containing the following files:

```
your_submission_folder/
├── model.pth                    # REQUIRED: Your trained model
├── info.json                    # REQUIRED: Model information
├── model.py                     # REQUIRED: Architecture definition
└── transforms.py                # OPTIONAL: Custom test transforms
```

Required Files

1. `model.pth` (REQUIRED)

Your trained model weights. For saving your model weights use this:

```
import torch

# Save only the weights
torch.save(model.state_dict(), 'model.pth')
```

2. `info.json` (REQUIRED)

A JSON file containing information about your model and submission.

Minimal example:

```
{
  "group_id": "A",
}
```

Complete example with all optional fields:

```
{
  "group_id": "A",
  "architecture": "resnet50_custom",
  "description": "ResNet50 with custom classifier head and aggressive data
augmentation",
  "training_details": {
    "epochs": 30,
    "learning_rate": 0.001,
    "optimizer": "Adam",
    "batch_size": 32,
    "loss_function": "CrossEntropyLoss",
    "data_augmentation": "horizontal flip, rotation, color jitter",
    "validation_accuracy": 0.92
  }
}
```

Field descriptions:

Field	Required	Description
`group_id`	☑ Yes	Your gruop ID
`architecture`	❌ No	Model architecture name (e.g., "resnet50", "custom")
`description`	❌ No	Brief description of your approach
`training_details`	❌ No	Any relevant training information

---

3. `model.py` (REQUIRED)

Template:

```
import torch.nn as nn
from torchvision import models

def get_model(num_classes=2):
    """
    This function must return your model architecture.

    Args:
        num_classes (int): Number of output classes (always 2 for this
competition)

    Returns:
        model: Your PyTorch model
    """
    # Example 1: Standard architecture
```

```
    model = models.resnet50(pretrained=False)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model
```

Example: Modified standard architecture

```
import torch.nn as nn
from torchvision import models

def get_model(num_classes=2):
    # Load base model
    model = models.resnet50(pretrained=False)

    # Custom classifier head
    in_features = model.fc.in_features
    model.fc = nn.Sequential(
        nn.Dropout(0.3),
        nn.Linear(in_features, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Dropout(0.3),
        nn.Linear(512, num_classes)
    )

    return model
```

Example: Completely custom architecture

```
import torch.nn as nn

class CustomGlaucomaCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(CustomGlaucomaCNN, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.classifier = nn.Sequential(
            nn.Linear(256 * 28 * 28, 512),
```

```
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

def get_model(num_classes=2):
    return CustomGlaucomaCNN(num_classes=num_classes)
```

Optional Files

4. `transforms.py` (OPTIONAL)

Define custom preprocessing/transforms for test images. If not provided, default ImageNet transforms will be used.

Template:

```
from torchvision import transforms

def get_test_transform():
    """
    This function must return the transforms to apply to test images.

    IMPORTANT: These should match the transforms you used during training
    (without data augmentation).

    Returns:
        transform: torchvision.transforms.Compose object
    """
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
```

Example: Custom normalization

```python
from torchvision import transforms

def get_test_transform():
    # If you trained with different normalization values
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.5, 0.5, 0.5],  # Your custom values
            std=[0.5, 0.5, 0.5]
        )
    ])
```

Example: Different input size

```python
from torchvision import transforms

def get_test_transform():
    # If your model expects 256x256 images
    return transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
```

☑ Submission Checklist

Before submitting, verify:

- ☐ Your folder contains `model.pth`, `model.py` and `info.json` (must be named exactly in this way, case sensitive)
- ☐ Your `group_id` is correct in `info.json`
- ☐ If you used custom preprocessing, you've included `transforms.py`
- ☐ All Python files are valid and can be imported without errors
- ☐ Your model works with 2 output classes (normal/glaucoma)
- ☐ You've tested loading your model locally before submission

---

📊 Evaluation Metrics

Your model will be evaluated on a **secret test set** using the following metrics:

- **Balanced Accuracy**: Overall correctness

- **Accuracy**: Overall correctness
- **Precision**: Of positive predictions (glaucoma), how many are correct?
- **Recall (Sensitivity)**: Of actual glaucoma cases, how many did we catch?
- **Specificity**: Of actual normal cases, how many did we correctly identify?
- **F1 Score**: Harmonic mean of precision and recall
- **AUC-ROC**: Area under the ROC curve

Ranking: Final ranking will be based primarily on **Balanced Accuracy**, with F1-Score as tiebreaker.

Example Submission Scenarios

Scenario 1: Using ResNet50 (Standard)

```
my_submission/
├── model.pth                # Saved with torch.save(model.state_dict(), ...)
├── info.json
└── model.py                 # get_model() returns ResNet50
```

Scenario 2: Using EfficientNet with custom transforms

```
my_submission/
├── model.pth                # Saved with torch.save(model.state_dict(), ...)
├── info.json
├── model.py                 # get_model() returns EfficientNet
└── transforms.py            # Custom preprocessing
```

Scenario 3: Completely custom architecture

```
my_submission/
├── model.pth                # Saved with torch.save(model.state_dict(), ...)
├── info.json
├── model.py                 # Custom CNN class + get_model()
└── transforms.py            # Custom preprocessing
```

Need Help?

If you encounter any issues:

1. Check this document thoroughly
2. Verify all required files are present
3. Test loading your model locally
4. Contact the instructor with:
    - Your submission folder structure

- The exact error message (if any)
- Your `info.json` content

Good luck with the competition! 🚀

Remember: The goal is not just to achieve high accuracy, but to understand **why** your model works and what it's learning from the retinal images.