## Initial setup

↳ 3 cells hidden

## Data pre-processing

↳ 8 cells hidden

## A/B Testing with Baseline Model

↳ 18 cells hidden

## Best current model is that with augmentation and using the full image

```
def set_seed(seed: int = 42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
seed=42
set_seed(42)

SAVE_DIR = "/content/bike_lane_runs"
os.makedirs(SAVE_DIR, exist_ok=True)
print("Saving checkpoints/configs to:", SAVE_DIR)
```

```
Saving checkpoints/configs to: /content/bike_lane_runs
```

```
# updated old function
```

```
def train_one_model(
    name,
    use_lane_crop,
    max_epochs=15,
    lr=1e-4,
    batch_size=8,
    save_dir=SAVE_DIR,
    seed=seed,
    min_delta=1e-3,       # min improvement in val_acc to count as be
    patience=3            # stop if no meaningful improvement after 3
):
    print(f"Training model: {name} (use_lane_crop={use_lane_crop})")

    set_seed(seed)

    train_loader, val_loader, test_loader = make_loaders(
        use_lane_crop, batch_size=batch_size
    )

    backbone = make_backbone()
    model = SegFormerBlockedClassifier(backbone, num_classes=2).to(de

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr)

    train_losses = []
    val_accs = []

    best_val_acc = -1.0
    best_epoch = -1
    epochs_no_improve = 0

    os.makedirs(save_dir, exist_ok=True)
    best_ckpt_path = os.path.join(save_dir, f"{name}_best.pt")

    for epoch in range(1, max_epochs + 1):
        model.train()
        running_loss = 0.0

        for pixel_values, labels in train_loader:
            pixel_values = pixel_values.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            logits = model(pixel_values)
```

```
            loss = criterion(logits, labels)
            loss.backward()
            optimizer.step()

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

        train_loss = running_loss / len(train_loader.dataset)
        val_acc, _, _ = evaluate(model, val_loader)

        train_losses.append(train_loss)
        val_accs.append(val_acc)

        print(f"[{name}] Epoch {epoch}/{max_epochs} - "
              f"train_loss: {train_loss:.4f} - val_acc: {val_acc:.4f}"

        # check for improvement
        if val_acc > best_val_acc + min_delta:
            best_val_acc = val_acc
            best_epoch = epoch
            epochs_no_improve = 0

            torch.save(
                {
                    "model_state_dict": model.state_dict(),
                    "optimizer_state_dict": optimizer.state_dict(),
                    "epoch": best_epoch,
                    "val_acc": best_val_acc,
                    "seed": seed,
                    "use_lane_crop": use_lane_crop,
                    "lr": lr,
                    "batch_size": batch_size,
                    "min_delta": min_delta,
                    "patience": patience,
                },
                best_ckpt_path,
            )
            print(f"  ↳ New best val_acc={best_val_acc:.4f} at epoch
                  f"saved to {best_ckpt_path}")
        else:
            epochs_no_improve += 1
            print(f"  ↳ No significant improvement (Δval_acc={val_acc
                  f"epochs_no_improve={epochs_no_improve}/{patience}"

        # early stopping?
        if epochs_no_improve >= patience:
```

```
            print(f"\nEarly stopping triggered at epoch {epoch} "
                  f"(no improvement > {min_delta} for {patience} epocl
            break

    # plot
    epochs_ran = range(1, len(train_losses) + 1)

    plt.figure(figsize=(8, 4))
    plt.plot(epochs_ran, train_losses, marker="o")
    plt.title(f"{name} - Training Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(8, 4))
    plt.plot(epochs_ran, val_accs, marker="o")
    plt.title(f"{name} - Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.ylim(0, 1.0)
    plt.grid(True)
    plt.show()

    # best checkpoint
    print(f"\n[{name}] Loading best checkpoint from epoch {best_epoch}
          f"(val_acc={best_val_acc:.4f})")
    checkpoint = torch.load(best_ckpt_path, map_location=device)
    model.load_state_dict(checkpoint["model_state_dict"])

    # test eval
    test_acc, test_cm, test_report = evaluate(model, test_loader)
    print(f"\n[{name}] TEST accuracy: {test_acc:.4f}")
    print(f"[{name}] TEST confusion matrix (rows = true, cols = pred;
    print(test_cm)
    print(f"\n[{name}] TEST classification report:")
    print(test_report)

    config = {
        "name": name,
        "seed": seed,
        "max_epochs": max_epochs,
        "lr": lr,
        "batch_size": batch_size,
        "use_lane_crop": use_lane_crop,
```

```
        "augment": True,
        "backbone": "nvidia/segformer-b0-finetuned-ade-512-512",
        "best_epoch": best_epoch,
        "best_val_acc": float(best_val_acc),
        "test_accuracy": float(test_acc),
        "min_delta": float(min_delta),
        "patience": patience,
    }

    config_path = os.path.join(save_dir, f"{name}_config.json")
    with open(config_path, "w") as f:
        json.dump(config, f, indent=2)
    print(f"\n[{name}] Saved run config to {config_path}")

    history = {
        "train_losses": train_losses,
        "val_accs": val_accs,
    }

    return model, (test_acc, test_cm, test_report, history)
```

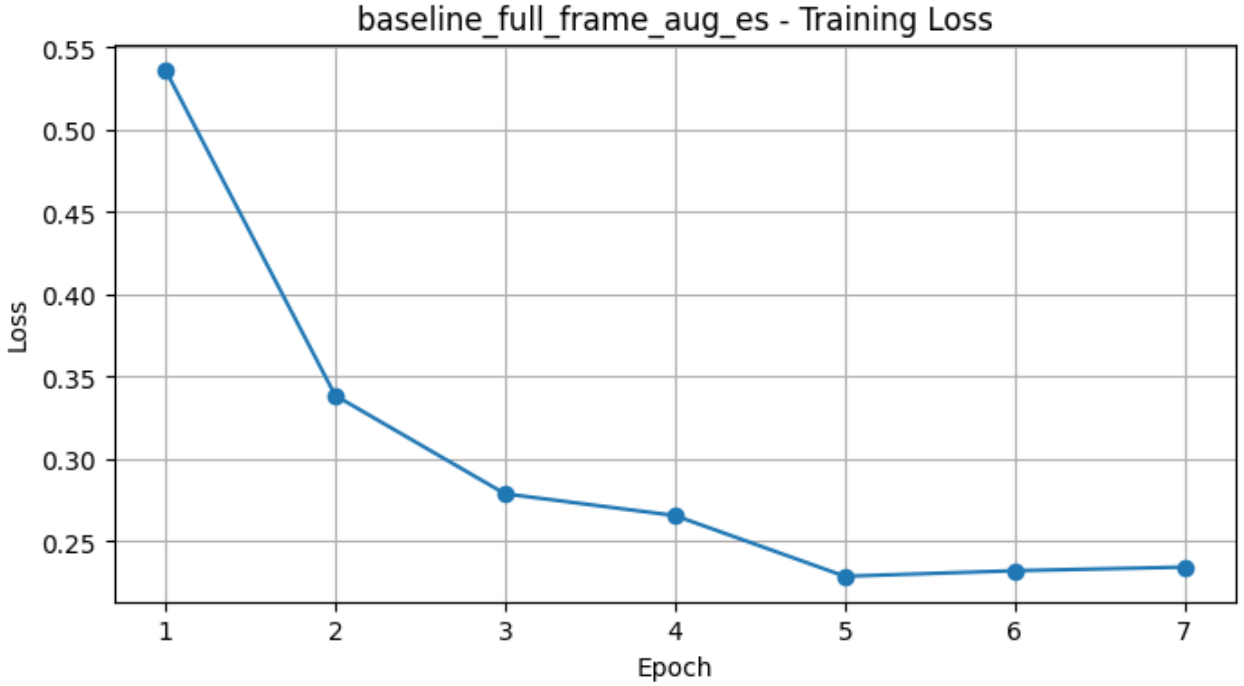
In this cell, I train the baseline segformers classifier on full-frame images w data augmentation and early stopping to get a clean model run:

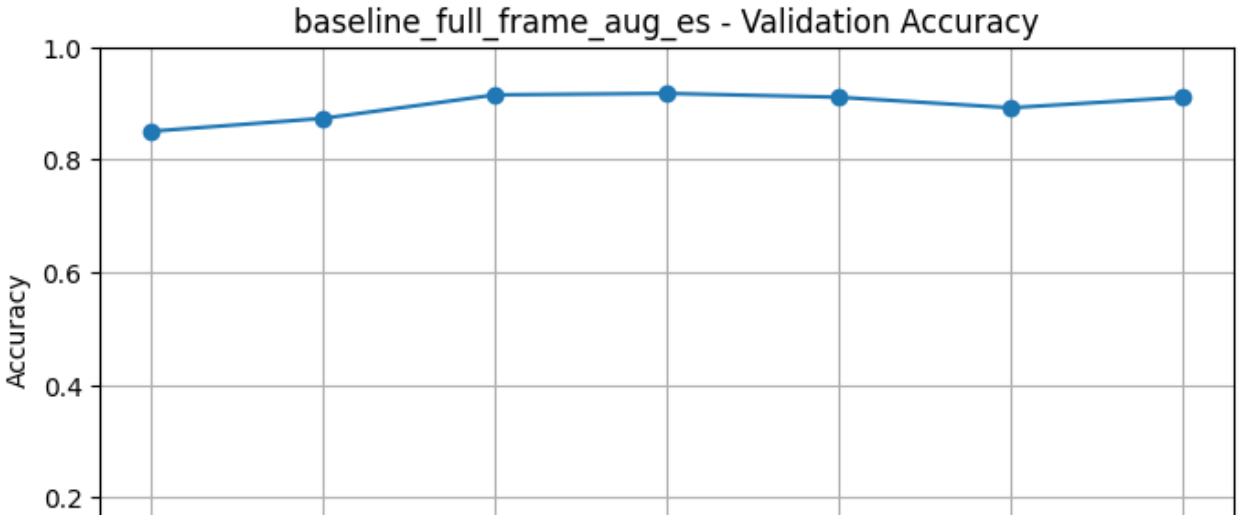
- min_delta = 0.001 (val accuracy must improve by at least 0.001)
- patience = 3 (stop if no such improvement for 3 consecutive epochs)

```
baseline_model, baseline_metrics = train_one_model(
    name="baseline_full_frame_aug_es",
    use_lane_crop=False,              # using full images
    max_epochs=15,
    lr=1e-4,
    batch_size=8,
    save_dir=SAVE_DIR,
    seed=seed,
    min_delta=1e-3,                   # require at least 0.001 val_acc
    patience=3                        # stop after 3 epochs with no su
)
```

```
Training model: baseline_full_frame_aug_es (use_lane_crop=False)
/content/split_camera/train: found 2739 images (use_lane_crop=False, au
/content/split_camera/val: found 1175 images (use_lane_crop=False, augm
/content/split_camera/test: found 38 images (use_lane_crop=False, augme
```

```
[baseline_full_frame_aug_es] Epoch 1/15 - train_loss: 0.5365 - val_acc:
  ↳ New best val_acc=0.8502 at epoch 1, saved to /content/bike_lane_run
[baseline_full_frame_aug_es] Epoch 2/15 - train_loss: 0.3386 - val_acc:
  ↳ New best val_acc=0.8732 at epoch 2, saved to /content/bike_lane_run
[baseline_full_frame_aug_es] Epoch 3/15 - train_loss: 0.2789 - val_acc:
  ↳ New best val_acc=0.9149 at epoch 3, saved to /content/bike_lane_run
[baseline_full_frame_aug_es] Epoch 4/15 - train_loss: 0.2657 - val_acc:
  ↳ New best val_acc=0.9174 at epoch 4, saved to /content/bike_lane_run
[baseline_full_frame_aug_es] Epoch 5/15 - train_loss: 0.2288 - val_acc:
  ↳ No significant improvement (Δval_acc=-0.0068), epochs_no_improve=1/
[baseline_full_frame_aug_es] Epoch 6/15 - train_loss: 0.2322 - val_acc:
  ↳ No significant improvement (Δval_acc=-0.0255), epochs_no_improve=2/
[baseline_full_frame_aug_es] Epoch 7/15 - train_loss: 0.2344 - val_acc:
  ↳ No significant improvement (Δval_acc=-0.0068), epochs_no_improve=3/

Early stopping triggered at epoch 7 (no improvement > 0.001 for 3 epoch
```



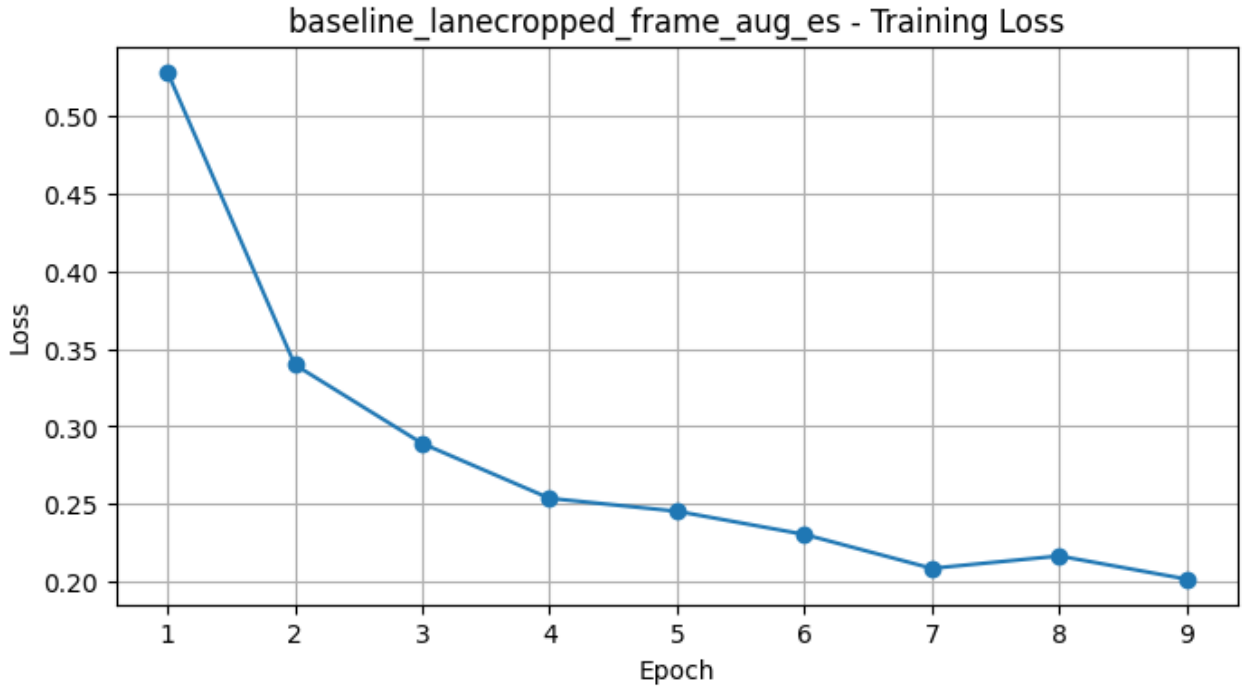



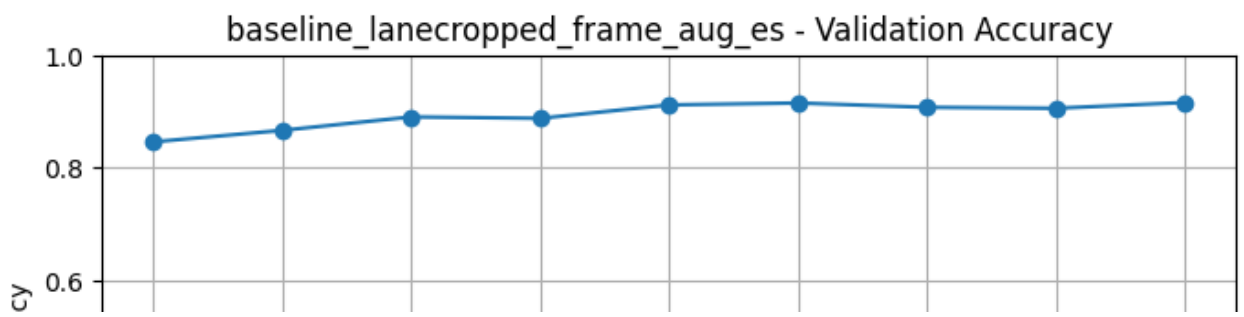

```
[baseline_full_frame_aug_es] Loading best checkpoint from epoch 4 (val_

[baseline_full_frame_aug_es] TEST accuracy: 0.3947
[baseline_full_frame_aug_es] TEST confusion matrix (rows = true, cols =
[[ 4 23]
 [ 0 11]]

[baseline_full_frame_aug_es] TEST classification report:
              precision    recall  f1-score   support

  notblocked      1.000     0.148     0.258        27
     blocked      0.324     1.000     0.489        11

    accuracy                         0.395        38
   macro avg      0.662     0.574     0.373        38
weighted avg      0.804     0.395     0.325        38


[baseline_full_frame_aug_es] Saved run config to /content/bike_lane_run
```

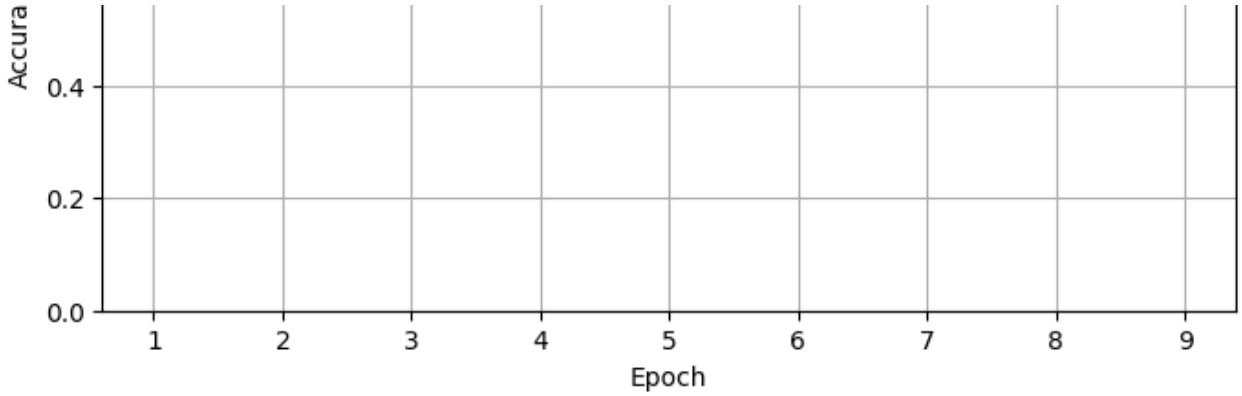
```
baseline_modelC, baseline_metricsC = train_one_model(
    name="baseline_lanecropped_frame_aug_es",
    use_lane_crop=True,               # using full images
    max_epochs=15,
    lr=1e-4,
    batch_size=8,
    save_dir=SAVE_DIR,
    seed=seed,
    min_delta=1e-3,                    # require at least 0.001 val_a
    patience=3                         # stop after 3 epochs with no
)
```

```
Training model: baseline_lanecropped_frame_aug_es (use_lane_crop=True)
/content/split_camera/train: found 2739 images (use_lane_crop=True, aug
```

```
/content/split_camera/val: found 1175 images (use_lane_crop=True, augme
/content/split_camera/test: found 38 images (use_lane_crop=True, augmen
[baseline_lanecropped_frame_aug_es] Epoch 1/15 - train_loss: 0.5284 - v
  ↳ New best val_acc=0.8460 at epoch 1, saved to /content/bike_lane_run
[baseline_lanecropped_frame_aug_es] Epoch 2/15 - train_loss: 0.3397 - v
  ↳ New best val_acc=0.8664 at epoch 2, saved to /content/bike_lane_run
[baseline_lanecropped_frame_aug_es] Epoch 3/15 - train_loss: 0.2890 - v
  ↳ New best val_acc=0.8902 at epoch 3, saved to /content/bike_lane_run
[baseline_lanecropped_frame_aug_es] Epoch 4/15 - train_loss: 0.2536 - v
  ↳ No significant improvement (Δval_acc=-0.0026), epochs_no_improve=1/
[baseline_lanecropped_frame_aug_es] Epoch 5/15 - train_loss: 0.2453 - v
  ↳ New best val_acc=0.9115 at epoch 5, saved to /content/bike_lane_run
[baseline_lanecropped_frame_aug_es] Epoch 6/15 - train_loss: 0.2304 - v
  ↳ New best val_acc=0.9149 at epoch 6, saved to /content/bike_lane_run
[baseline_lanecropped_frame_aug_es] Epoch 7/15 - train_loss: 0.2085 - v
  ↳ No significant improvement (Δval_acc=-0.0077), epochs_no_improve=1/
[baseline_lanecropped_frame_aug_es] Epoch 8/15 - train_loss: 0.2165 - v
  ↳ No significant improvement (Δval_acc=-0.0094), epochs_no_improve=2/
[baseline_lanecropped_frame_aug_es] Epoch 9/15 - train_loss: 0.2016 - v
  ↳ No significant improvement (Δval_acc=0.0009), epochs_no_improve=3/3

Early stopping triggered at epoch 9 (no improvement > 0.001 for 3 epoch
```






```
[baseline_lanecropped_frame_aug_es] Loading best checkpoint from epoch

[baseline_lanecropped_frame_aug_es] TEST accuracy: 0.2895
[baseline_lanecropped_frame_aug_es] TEST confusion matrix (rows = true,
[[ 0 27]
 [ 0 11]]

[baseline_lanecropped_frame_aug_es] TEST classification report:
              precision    recall  f1-score   support

  notblocked      0.000     0.000     0.000        27
     blocked      0.289     1.000     0.449        11

    accuracy                         0.289        38
   macro avg      0.145     0.500     0.224        38
weighted avg      0.084     0.289     0.130        38


[baseline_lanecropped_frame_aug_es] Saved run config to /content/bike_l
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
```

During training, we see early stopping is triggered at epoch 7, meaning that validation accuracy stopped improving meaningfully after epoch 4. Epoch 4's validation accuracy = 0.9174

When I evaluate it on the held-out cam68 test set,

- Test accuracy: 0.3947
- For notblocked (27 images):
  - 4 correctly predicted as notblocked
  - 23 incorrectly predicted as blocked
  - → very low recall (0.148), but precision = 1.0 (when the model says "notblocked", it's always correct, just very rare)
- For blocked (11 images):
  - 11 correctly predicted as blocked
  - 0 missed
  - → recall = 1.0, but precision is only 0.324 (many notblocked images are being flagged as blocked)

Overall, this operating point is extremely conservative in the sense of rarely missing a blocked lane (100% blocked recall), but it generates many false alarms on notblocked images, leading to low overall accuracy (~ 39.5%). This motivates the threshold-tuning step that follows.

# Tuned classifier

```
# use_lane_crop=False bc cropped was worse
train_loader, val_loader, test_loader = make_loaders(
    use_lane_crop=False,
    batch_size=8
)
```

```
/content/split_camera/train: found 2739 images (use_lane_crop=False, a
/content/split_camera/val: found 1175 images (use_lane_crop=False, augm
/content/split_camera/test: found 38 images (use_lane_crop=False, augme
```

```
# collect p(blocked) and true labels (probabilities)
def get_probs_and_labels(model, loader):
    model.eval()
    all_probs = []
    all_labels = []

    with torch.no_grad():
        for pixel_values, labels in loader:
            pixel_values = pixel_values.to(device)
            labels = labels.to(device)

            logits = model(pixel_values)               # (B, 2)
            probs = torch.softmax(logits, dim=1)       # (B, 2)
            p_blocked = probs[:, 1]                    # prob of class

            all_probs.extend(p_blocked.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_probs), np.array(all_labels)

val_probs, val_labels   = get_probs_and_labels(baseline_model, val_loa
test_probs, test_labels = get_probs_and_labels(baseline_model, test_lo

print("Val shape:", val_probs.shape, val_labels.shape)
print("Test shape:", test_probs.shape, test_labels.shape)

# val_probs[i] = model's probability that image i is blocked
# val_labels[i] = 0 (notblocked) or 1 (blocked)
```

```
Val shape: (1175,) (1175,)
Test shape: (38,) (38,)
```

```
def sweep_thresholds(probs, labels, num_steps=101):
    thresholds = np.linspace(0.0, 1.0, num_steps)
    rows = []

    for t in thresholds:
        preds = (probs >= t).astype(int)  # 1 = blocked, 0 = notblocke

        # metrics for both classes
        precision, recall, f1, _ = precision_recall_fscore_support(
            labels,
            preds,
            labels=[0, 1],   # [notblocked, blocked]
```

```
            zero_division=0
        )

        # unpack
        prec_not, rec_not, f1_not = precision[0], recall[0], f1[0]
        prec_blk, rec_blk, f1_blk = precision[1], recall[1], f1[1]

        rows.append({
            "threshold": t,
            "prec_blocked": prec_blk,
            "rec_blocked": rec_blk,
            "f1_blocked": f1_blk,
            "prec_notblocked": prec_not,
            "rec_notblocked": rec_not,
            "f1_notblocked": f1_not,
        })

    return rows

val_sweep = sweep_thresholds(val_probs, val_labels, num_steps=101)
len(val_sweep), val_sweep[:5]
val_df = pd.DataFrame(val_sweep)
val_df.head()
```

| | threshold | prec_blocked | rec_blocked | f1_blocked | prec_notblocked | re |
|---|---|---|---|---|---|---|
| **0** | 0.00 | 0.566809 | 1.000000 | 0.723520 | 0.000000 | |
| **1** | 0.01 | 0.566809 | 1.000000 | 0.723520 | 0.000000 | |
| **2** | 0.02 | 0.573643 | 1.000000 | 0.729064 | 1.000000 | |
| **3** | 0.03 | 0.592000 | 1.000000 | 0.743719 | 1.000000 | |
| **4** | 0.04 | 0.630930 | 0.998498 | 0.773256 | 0.991736 | |

Next steps: Generate code with `val_df` New interactive sheet

Idea 1: pick threshold based on not wanting to miss blockages (high recall), but still trying to keep reasonable precision

```
# filter to thresholds where recall for blocked is >= 0.9
```

```
candidates = val_df[val_df["rec_blocked"] >= 0.9].copy()

if len(candidates) == 0:
    print("No thresholds reach recall >= 0.9 for blocked on val set."
    # fall back to just the best F1 overall
    best_row = val_df.iloc[val_df["f1_blocked"].idxmax()]
else:
    # among those, pick the one with highest f1_blocked
    best_row = candidates.iloc[candidates["f1_blocked"].idxmax()]

best_threshold = float(best_row["threshold"])
print("Chosen threshold (p_blocked >= t → blocked):", best_threshold)
print(best_row)

# use best_threshold on the test probabilities
def evaluate_with_threshold(probs, labels, threshold):
    preds = (probs >= threshold).astype(int)

    acc = (preds == labels).mean()
    cm = confusion_matrix(labels, preds, labels=[0, 1])
    report = classification_report(
        labels,
        preds,
        labels=[0, 1],
        target_names=["notblocked", "blocked"],
        digits=3
    )
    return acc, cm, report

test_acc_t, test_cm_t, test_report_t = evaluate_with_threshold(
    test_probs, test_labels, best_threshold
)

print(f"Test accuracy at threshold={best_threshold:.3f}: {test_acc_t:
print("Confusion matrix (rows = true, cols = pred; 0=notblocked,1=blo
print(test_cm_t)
print("\nClassification report:")
print(test_report_t)
```

```
Chosen threshold (p_blocked >= t → blocked): 0.48
threshold          0.480000
prec_blocked       0.917889
rec_blocked        0.939940
f1_blocked         0.928783
prec_notblocked    0.918864
```

```
rec_notblocked     0.889980
f1_notblocked      0.904192
balanced_acc       0.914960
Name: 48, dtype: float64
Test accuracy at threshold=0.480: 0.3684
Confusion matrix (rows = true, cols = pred; 0=notblocked,1=blocked):
[[ 3 24]
 [ 0 11]]

Classification report:
              precision    recall  f1-score   support

  notblocked      1.000     0.111     0.200        27
     blocked      0.314     1.000     0.478        11

    accuracy                          0.368        38
   macro avg      0.657     0.556     0.339        38
weighted avg      0.802     0.368     0.281        38
```

Idea 2: bc the recall-only approach did worse than the baseline, going to try and change the threshold metric to balanced accuracy ≈ average of class recalls

```
val_df["balanced_acc"] = 0.5 * (val_df["rec_blocked"] + val_df["rec_not

best_row_bal = val_df.iloc[val_df["balanced_acc"].idxmax()]
t_bal = float(best_row_bal["threshold"])
print("Best balanced-accuracy threshold on val:", t_bal)
print(best_row_bal)

test_acc_bal, test_cm_bal, test_report_bal = evaluate_with_threshold(
    test_probs, test_labels, t_bal
)
print(f"\nTest accuracy at threshold={t_bal:.3f}: {test_acc_bal:.4f}")
print("Confusion matrix:")
print(test_cm_bal)
print("\nClassification report:")
print(test_report_bal)
```

```
Best balanced-accuracy threshold on val: 0.75
threshold          0.750000
prec_blocked       0.956311
rec_blocked        0.887387
f1_blocked         0.920561
prec_notblocked    0.865350
rec_notblocked     0.946955
f1_notblocked      0.904315
balanced_acc       0.917171
Name: 75, dtype: float64

Test accuracy at threshold=0.750: 0.6316
Confusion matrix:
[[15 12]
 [ 2  9]]

Classification report:
              precision    recall  f1-score   support

  notblocked      0.882     0.556     0.682        27
     blocked      0.429     0.818     0.562        11

    accuracy                          0.632        38
   macro avg      0.655     0.687     0.622        38
weighted avg      0.751     0.632     0.647        38
```

```
# t=0.5 threshold for comparison
row_05 = val_df.iloc[(val_df["threshold"] - 0.5).abs().idxmin()]
print("Metrics near t=0.5 on val:")
print(row_05)
```

```
Metrics near t=0.5 on val:
threshold          0.500000
prec_blocked       0.920236
rec_blocked        0.935435
f1_blocked         0.927774
prec_notblocked    0.913655
rec_notblocked     0.893910
f1_notblocked      0.903674
balanced_acc       0.914673
Name: 50, dtype: float64
```

# Understand why model making mistakes at 0.75 threshold - step 3

```
# using best threshold t_bal from step 2
operating_threshold = float(t_bal)
print("Using best operating threshold (p_blocked):", operating_thresh

test_ds = test_loader.dataset
print("Test set size:", len(test_ds))

label_id2name = {v: k for k, v in label_map.items()}

# run model on test set and categorize TP / TN / FP / FN
def analyze_test_set(model, dataset, threshold):
    model.eval()
    records = []

    with torch.no_grad():
        for i in range(len(dataset)):
            # dataset.samples should hold (filepath, class_name)
            img_path, label_name = dataset.samples[i]
            pixel_values, label_tensor = dataset[i]

            pixel_values = pixel_values.unsqueeze(0).to(device)  # (1,
```

```
            true_id = label_tensor.item()

            logits = model(pixel_values)                # (1, 2)
            probs = torch.softmax(logits, dim=1)        # (1, 2)
            p_blocked = probs[0, 1].item()              # prob of cl

            pred_id = int(p_blocked >= threshold)       # 1 = blocke

            # categorize
            if true_id == 1 and pred_id == 1:
                err_type = "TP"   # blocked correctly detected
            elif true_id == 0 and pred_id == 0:
                err_type = "TN"   # notblocked correctly detected
            elif true_id == 0 and pred_id == 1:
                err_type = "FP"   # predicted blocked, actually notbl
            else:  # true_id == 1 and pred_id == 0
                err_type = "FN"   # missed blockage

            records.append({
                "idx": i,
                "path": str(img_path),
                "true_id": true_id,
                "true_name": label_id2name[true_id],
                "pred_id": pred_id,
                "pred_name": label_id2name[pred_id],
                "p_blocked": p_blocked,
                "type": err_type,
            })

    return records

records = analyze_test_set(baseline_model, test_ds, operating_thresho

TP = [r for r in records if r["type"] == "TP"]
TN = [r for r in records if r["type"] == "TN"]
FP = [r for r in records if r["type"] == "FP"]
FN = [r for r in records if r["type"] == "FN"]

print("\nCounts at threshold", operating_threshold)
print("  TP:", len(TP), "TN:", len(TN), "FP:", len(FP), "FN:", len(FN
```

```
Using best operating threshold (p_blocked): 0.75
Test set size: 38

Counts at threshold 0.75
```

```
TP: 9 TN: 15 FP: 12 FN: 2
```

```
# helpers for summarziing and visualziing misclassified
def print_examples(examples, title, max_n=20):
    print(title, f"(showing up to {max_n})")
    for i, r in enumerate(examples[:max_n]):
        print(f"{i+1:2d}. {r['path']}")
        print(f"    true: {r['true_name']:<10s}  "
              f"pred: {r['pred_name']:<10s}  "
              f"p_blocked={r['p_blocked']:.3f}")
    print()

print_examples(FP, "False Positives (predicted blocked, actually notblo
print_examples(FN, "False Negatives (missed blocked)", max_n=50)

def show_examples(examples, title, n=6):
    n = min(n, len(examples))
    if n == 0:
        print(f"No examples for {title}")
        return

    cols = 3
    rows = (n + cols - 1) // cols

    plt.figure(figsize=(5*cols, 4*rows))
    for i in range(n):
        r = examples[i]
        img = Image.open(r["path"]).convert("RGB")

        plt.subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.axis("off")
        plt.title(
            f"true: {r['true_name']}\n"
            f"pred: {r['pred_name']}\n"
            f"p_blocked={r['p_blocked']:.2f}",
            fontsize=9
        )

    plt.suptitle(title, fontsize=14)
    plt.tight_layout()
    plt.show()

# show a few of each type of mistake
show_examples(FP, "False Positives (notblocked → blocked)", n=8)
```

```
show_examples(FN, "False Negatives (blocked → notblocked)", n=8)
```

```
False Positives (predicted blocked, actually notblocked) (showing up to
 1. /content/split_camera/test/notblocked/2016-09-16 151137 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.811
 2. /content/split_camera/test/notblocked/2016-09-16 152433 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.828
 3. /content/split_camera/test/notblocked/2016-09-16 151036 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.779
 4. /content/split_camera/test/notblocked/2016-09-16 151238 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.827
 5. /content/split_camera/test/notblocked/2016-09-16 151726 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.821
 6. /content/split_camera/test/notblocked/2016-09-16 152332 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.787
 7. /content/split_camera/test/notblocked/2016-09-16 155832 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.932
 8. /content/split_camera/test/notblocked/2016-09-16 155731 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.884
 9. /content/split_camera/test/notblocked/2016-09-16 155125 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.839
10. /content/split_camera/test/notblocked/2016-09-16 151827 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.771
11. /content/split_camera/test/notblocked/2016-09-16 153105 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.942
12. /content/split_camera/test/notblocked/2016-09-16 152635 cam68.png
    true: notblocked  pred: blocked     p_blocked=0.894

False Negatives (missed blocked) (showing up to 50)
 1. /content/split_camera/test/blocked/2016-09-16 153610 cam68.png
    true: blocked     pred: notblocked  p_blocked=0.722
 2. /content/split_camera/test/blocked/2016-09-16 153509 cam68.png
    true: blocked     pred: notblocked  p_blocked=0.691
```

true: notblocked
pred: blocked
p_blocked=0.93


true: notblocked
pred: blocked
p_blocked=0.88


False Neg

true: blocked
pred: notblocked
p_blocked=0.72


# Takeaways

- Model is p good at catching blocked (9/11 --> recall = 0.82)
- We have a high amount of false alarms due to the fact that the lane is very slightly blocked by a truck, but this doesn't count as fully obstructed for a bike to not be able to pass through at all. This isn't bad in my opinion though, because it just further prioritizes biker saftey.
- Model assigns high blocked probabilities (p_blocked from 0.78–0.94), showing it may have learned a strong association between visual conetxt cues and the blocked label in training even when the true label is notblocked.
- Only 2 legitimately blocked imgs are missed and both are borderline cases (v close to being predicted as blocked w probabilities 0.722 and 0.691).

```
Start coding or generate with AI.
```