TRANSFER LEARNING (RESEARCH)

- A method to use pre-existing ML models as a starting point for our own usecase and dataset. Instead of training models from scratch we use a well- trained model like Resnet18 and ImageNet to adapt to our task.
- Transfer Learning helps to preserve time as we do not need to build a model from scratch as it requires huge amount of resources.
- Steps in transfer learning:
  1) Choose a pre-trained model
  2) Freeze early layers (retain basic knowledge) and fine tune top layers.
  3) Replace top layers that matches our concern
  4) Train data

Some applications of Transfer learning are image classification, object classification, NLP, etc

PYTORCH

- Open-source deep learning framework
- Heavily used in neural networks, CNN's, RNN's
- Convolutional neural networks- specializes in image and video processing
- Recurrent neural networks- for text processing

Everything In pytorch is based on tensor operations. A tensor is a multi-dimensional matrix consisting of same type of data.

```
import torch

# torch.empty(size): uninitiallized
x = torch.empty(1) # scalar
print("empty(1):", x)
x = torch.empty(3) # vector
print("empty(3):",x)
x = torch.empty(2, 3) #matrix
print("empty(2,3):",x)
```

Torch.empty helps us create a tensor, depends on our use case how many dimension and size we need.

- Requires_grad arg: default set to false, need to set to true for calculation of gradients later on. (requires_grade= True).
- Numpy is a fundamental library that supports large multi-dimensional arrays and matrices, heavily used in deep learning.

TRANSFER LEARNING (RESEARCH)

How Pytorch actually works and the math behind it?

- Linear regression: the most basic algorithm in machine learning. Used to predict a value using linear graphs.
- Eg: wanting to predict weight of a person using linear approach **: y=mx+c**
- The goal of linear regression is to minimise mean squared error between predicted and actual values.

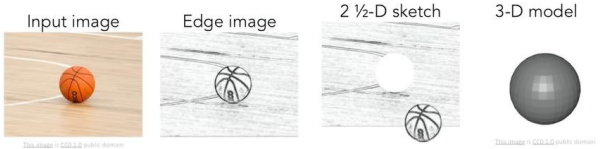
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}_i} - y_{\text{actual}_i})^2$$

  Lower the mse, better will be the model.
- Gradient descent: a method used to minimise loss function, ie finding the best value of m and c in the linear regression equation.
- Linear regression is the foundation for understanding how the models fit the data
- Gradient descent is almost how all the ML models learn.

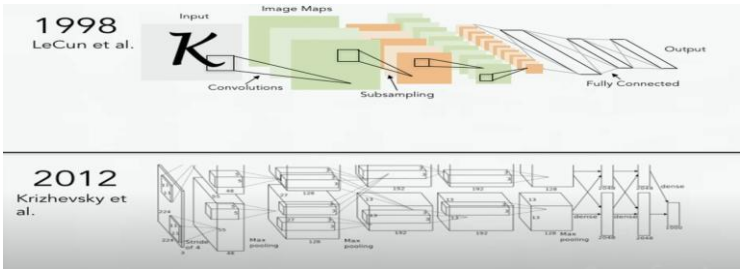
COMPUTER VISION

- From course cs231 we understand how computer vision evolved from history and how we tend to use it right now.
- Primal sketch

  Input image->edge image->2D->3D



- Problems of object recognition:
  1. Image classification
  2. Object detection
  3. Segmentation
  4. Facial recognition, scene understanding
- Convolutional neural networks found a breakthrough in the imageNet challenge helping it perform better than any other algorithm out there.
- Other name for CNN is deep learning using neural networks.



TRANSFER LEARNING (RESEARCH)

Why use **Google Colab**

1. Free Access to GPUs and TPUs

- You get free cloud-based access to powerful NVIDIA GPUs and TPUs.
- Ideal for training deep learning models like CNNs, transformers, etc.
- No need to buy an expensive GPU laptop/PC.

2. Runs in the Cloud

- You don't need to install Python, Jupyter, PyTorch, TensorFlow, etc., locally.
- Just log into your browser, and everything works.
- Your code runs on Google's servers, not your laptop.

3. Jupyter Notebook Interface

- Google Colab is built on Jupyter, so it's very beginner-friendly.
- You can:
    - Write & run Python code
    - Add markdown cells for notes
    - Visualize charts, graphs, and images inline

4. Great for Collaboration

- You can share your notebook just like Google Docs.
- Teammates can comment, suggest, or even run code with you.

5. Seamless with Google Drive

- Your notebooks are saved in your Google Drive.
- Easy to organize, access from anywhere, and back up

1. Linear Regression

What is it?

Linear regression is a supervised learning algorithm used for predicting a continuous value. It finds the best-fit straight line (or hyperplane in higher dimensions) through the data.

Equation:

$y=wx+b$

- x: input features
- w: weight/parameter
- b: bias/intercept
- y: predicted value

Goal:

Find the values of w and b that minimize the error between predictions and actual values.

TRANSFER LEARNING (RESEARCH)

---

2. Loss Function: Mean Squared Error (MSE)
The error is the difference between predicted and actual values.
MSE is the most commonly used loss for regression tasks:

- $$\mathcal{L}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
- - y_i: actual output
  - $\hat{y}_i$: predicted output
  - n: number of data points

---

3. Gradient Descent
Why Gradient Descent?
To minimize the loss function and find the optimal weights and bias.
Idea:
Take small steps in the opposite direction of the gradient of the loss function to reduce it.
Update Rule:

$$w = w - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w}$$

$$b = b - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b}$$

- α = learning rate (step size)
- $\frac{\partial \mathcal{L}}{\partial w}$ = gradient w.r.t. weight

---

How it Works Step-by-Step:

1. Start with random w and b.
2. Compute predictions using current w, b.
3. Calculate the loss using MSE.
4. Compute gradients of loss w.r.t. w and b.
5. Update w and b using gradient descent.
6. Repeat for multiple epochs until loss is minimized.

2. Activation Functions
Why do we need them?

- Without activation functions, a neural network would just be a linear model, no matter how many layers it has.
- Activation functions **introduce non-linearity**, allowing the network to learn more complex patterns.

---

1. Sigmoid Function
Equation:

Equation:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Output Range:

$$(0, 1)$$

TRANSFER LEARNING (RESEARCH)

Pros:

- Smooth gradient
- Good for binary classification (as output layer)

Cons:

- Vanishing gradient problem
- Outputs not centered around zero

2. Tanh (Hyperbolic Tangent)

Equation:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Output Range:

$$(-1, 1)$$

Pros:

- Zero-centered output
- Better than sigmoid in hidden layers

Cons:

- Still suffers from vanishing gradients

3. ReLU (Rectified Linear Unit)

Equation:

Equation:

$$f(x) = \max(0, x)$$

Output Range:

$$[0, \infty)$$

Pros:

- Simple and efficient
- Solves vanishing gradient issue (partially)

Cons:

- Can "die" during training if neurons stop updating (output = 0)

Function	Use Case
Sigmoid	Output layer in binary classification
Tanh	Hidden layers when data is zero-centered
ReLU	Default for hidden layers (fast and effective)

3. Loss Functions

What is a Loss Function?

A loss function measures how far off the model's predictions are from the actual values. It provides a **quantitative measure of error**, which is then minimized using **gradient descent**.

---

1. Mean Squared Error (MSE)

TRANSFER LEARNING (RESEARCH)

Use:

Used in **regression problems**.

Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i: actual value

$\hat{y}_i$: predicted value

n: number of samples

Pros:

- Simple and widely used
- Penalizes large errors

Cons:

- Sensitive to outliers

2. Cross-Entropy Loss

Use:

Used in **classification problems**.

Formula:

$$\mathcal{L} = -\sum_i y_i \log(\hat{y}_i)$$

y_i: true label (one-hot encoded)

$\hat{y}_i$: predicted probability

Pros:

- Ideal for probabilistic outputs
- Works well with Softmax activation

Cons:

- Output must be log-probabilities or raw logits

3. SVM Loss (Hinge Loss)

SVM LOSS:

Use:

Used in **Support Vector Machines (SVMs)**.

Formula:

$$\mathcal{L}(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

y: actual label (should be +1 or -1)

$f(x)$: predicted score

Pros:

- Focuses on margin maximization

TRANSFER LEARNING (RESEARCH)

- Works well for clear decision boundaries

Cons:

- Doesn't output probabilities

Loss Function	Task Type	Use With
MSE	Regression	Linear regression, DNN
Cross-Entropy	Classification	Softmax output
Hinge (SVM)	Classification	SVM models

1. Forward Pass

Goal: Calculate the output (prediction) of the model given an input.

How It Works:

- Data flows **forward** through each layer of the network.
- Each layer applies:
    - A linear transformation: $z=wx+b$
    - An activation function (e.g., ReLU, Sigmoid)

2. Loss Calculation

TRANSFER LEARNING (RESEARCH)

After the forward pass, we compare the prediction with the true value using a **loss function** like MSE or CrossEntropy.

3. Backward Pass (Backpropagation)

Goal: Compute gradients of the loss w.r.t. weights and biases using **chain rule** of calculus.

How It Works:

- The loss is propagated **backward** from the output layer to each layer.
- Gradients are computed for all parameters (weights & biases).

4. Parameter Update (Gradient Descent)

Once gradients are computed, we update parameters to reduce the loss:

```
optimizer.step()      # updates weights using gradients

optimizer.zero_grad() # clears old gradients before next backward pass
```

FULL EXAMPLE:

```
for epoch in range(n_epochs):

    output = model(inputs)            # forward pass

    loss = loss_fn(output, targets)   # compute loss

    loss.backward()                   # compute gradients

    optimizer.step()                  # update weights

optimizer.zero_grad()                 # reset gradients
```

What Backpropagation Gives Us

- It gives the **gradient** (slope) of the loss w.r.t. each parameter.
- This allows us to **know the direction** in which we need to change weights to reduce the error.

Input ➜ [Linear + Activation] ➜ Output ➜ Loss

▲ |

| ▼

Gradient ← Backward ← Loss Function

TRANSFER LEARNING (RESEARCH)

5. Batch Normalization

Why Do We Need It?

When training deep neural networks:

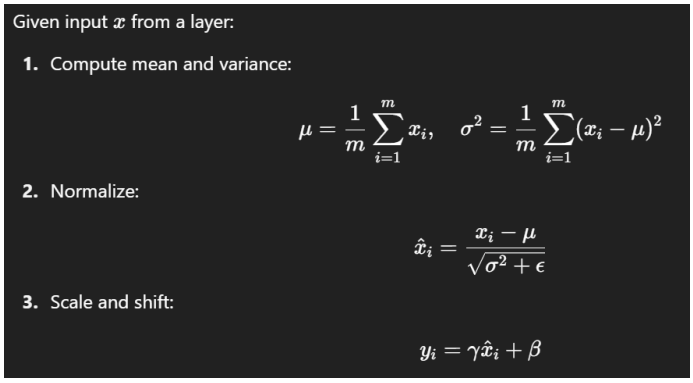
- The distribution of inputs to each layer changes during training.
- This slows down training and makes convergence harder (called **Internal Covariate Shift**).

Batch Normalization solves this by **normalizing** the inputs of each layer.

What It Does:

For each mini-batch during training, it:

1. **Normalizes** the inputs to have mean = 0 and variance = 1
2. **Scales and shifts** using learnable parameters γ\gammaγ (scale) and β\betaβ (shift)



Benefit	Description
Faster Training	Allows higher learning rates
Stability	Reduces exploding/vanishing gradients
Regularization	Acts like a mild regularizer, reduces need for dropout

BatchNorm helps make each layer learn **independently** from changes in previous layers' distributions — **making deep models easier to train**.

6. Transfer Learning

TRANSFER LEARNING (RESEARCH)

What is Transfer Learning?

Transfer Learning is the technique of **using a pre-trained model on a new but related task**.

Instead of training a model from scratch (which needs tons of data and time), we **leverage the knowledge** learned from a model trained on a large dataset (like ImageNet) and **fine-tune it** for our own task.

Common Use Case:

- Use a model trained on **ImageNet** (over 1 million images) for a smaller **image classification** task.

Part	**Role**
Base layers (Frozen)	Extract general features (edges, textures, shapes)
Final layer (Trainable)	Adapt to new task (like classifying cats vs dogs)
You Have...	Recommended Approach
Very small dataset	Freeze base model, train final layer
Medium dataset	Fine-tune top few layers
Large dataset, new task type	Train model from scratch

Key Insight:

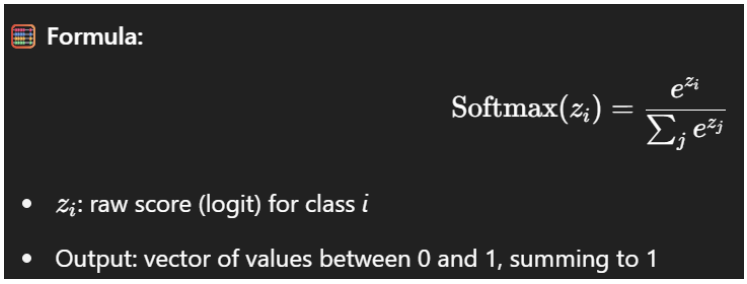
Transfer learning allows models to **generalize from one task to another**, just like how humans use prior knowledge to learn faster in new situations.

7. Softmax vs. SVM (Support Vector Machine)

🔸 What is Softmax?

Softmax is an **activation function** typically used in the final layer of a classification model to produce **probabilities** over classes.

TRANSFER LEARNING (RESEARCH)



Used With:

- **CrossEntropy Loss**

Pros:

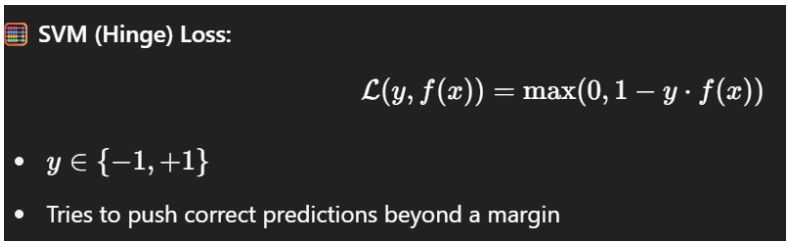
- Provides probabilities (confidence scores)
- Works well for multiclass classification

Cons:

- Can be overconfident
- Can struggle with small datasets unless regularized

🔸 **What is SVM?**

Support Vector Machines are **margin-based classifiers** that aim to **maximize the margin** between data points and the decision boundary.



Pros:

- Works well for small/medium datasets
- Focuses on support vectors → robust to outliers

Cons:

- Doesn't give probabilities
- Slower to train with large datasets

TRANSFER LEARNING (RESEARCH)

Feature	Softmax	SVM
Output	Probabilities	Raw scores (margins)
Loss Function	CrossEntropy Loss	Hinge Loss
Use Case	Neural Networks (deep learning)	Classic ML or shallow networks
Multiclass Support	Built-in	Requires one-vs-rest/one-vs-one
Optimization Goal	Maximize likelihood	Maximize margin
Confidence Output	Yes	No

Key Insight:

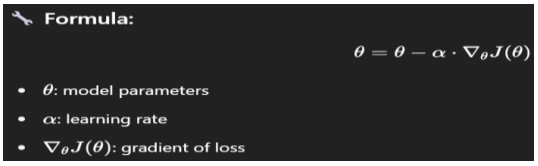
- Use **Softmax + CrossEntropy** in deep learning when you need **probabilities** and multiclass support.
- Use **SVM** when you're focusing on **maximizing margins**, especially in small datasets or classical ML setups.

What Is Optimization in ML?

Optimization refers to the method of **adjusting the model's parameters** (like weights and biases) to **minimize the loss function** — that is, improve predictions.

We use **gradients** (from backpropagation) to decide how to change weights. We apply this after gradients are computed after backprop.

A. **SGD (Stochastic Gradient Descent)**



Characteristics:

- Updates weights after each mini-batch
- Can be noisy → but helps escape local minima

Pros:

- Simple and memory efficient
- Good for large datasets

Cons:

- Needs careful tuning of learning rate
- Can get stuck or oscillate

B. **Momentum (Improved SGD)**

Formula:

$$v_t = \beta v_{t-1} + \alpha \nabla_\theta J(\theta) \quad \theta = \theta - v_t$$

- β: momentum coefficient (e.g., 0.9)

It adds velocity — so updates carry forward some momentum from the previous updates, smoothing out oscillations.

C. **RMSprop (Root Mean Square Propagation)**

Formula:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \quad \theta = \theta - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

- Keeps a moving average of squared gradients
- Scales learning rate for each parameter

Good for:

- non-stationary objectives

D. **Adam (Adaptive Moment Estimation)**

Combines Momentum + RMSprop

- Maintains both:
  - **Exponential moving average of gradients** (1st moment)
  - **Exponential moving average of squared gradients** (2nd moment)

TRANSFER LEARNING (RESEARCH)

Update Rule:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \theta = \theta - \alpha \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Pros:

- Requires less tuning
- Fast convergence
- Works well with sparse gradients

Optimizer	Best Use Case	Key Feature
SGD	Large datasets	Simplicity
Momentum	Faster convergence	Smooths update direction
RMSprop	RNNs, non-stationary objectives	Normalizes gradients
Adam	General-purpose optimizer	Combines RMSprop + Momentum

TRANSFER LEARNING (RESEARCH)

COMPUTER VISION

1. Image Classification

What is it?

Image classification is the most basic and widely used computer vision task. It involves assigning a single label to an entire image. For example, if you input an image of a dog, the model classifies it as “dog”.

You give it:

- An image (e.g., of a cat)

It gives you:

- A label: "cat"

Real-life Applications:

- Classifying X-ray images as normal or pneumonia.
- Detecting spam in memes (offensive, adult content).
- Animal or plant species classification.

Common Models

1. ResNet (Residual Network)
    - Solves the vanishing gradient problem.
    - Allows very deep networks by using skip connections (residual blocks).
    - Example: ResNet-50, ResNet-101.
2. VGG (Visual Geometry Group)
    - Uses very deep layers with small (3×3) filters.
    - Easy to implement, but heavy on computation.
3. DenseNet
    - Every layer is connected to all subsequent layers.
    - Helps reuse features and gradients more efficiently.
4. EfficientNet
    - Scales depth, width, and resolution systematically.
    - Very efficient and powerful for mobile and cloud.
5. Vision Transformer (ViT)
    - Uses self-attention mechanisms instead of convolution.
    - Performs great on large datasets.

TRANSFER LEARNING (RESEARCH)

Evaluation Metrics

1. Accuracy
    - Percentage of correct predictions over total predictions.
    - Good for balanced datasets.
2. Top-k Accuracy
    - Top-1 accuracy: correct label is the first prediction.
    - Top-5 accuracy: correct label is in the top 5 predicted labels.
3. Precision / Recall / F1-Score
    - Especially useful in imbalanced datasets (e.g., 90% cats, 10% dogs).
    - Precision: How many predicted cats are actually cats?
    - Recall: How many actual cats were detected?
    - F1: Harmonic mean of precision and recall.
4. Confusion Matrix
    - A grid showing where the model confused classes.
    - Helps diagnose specific mistakes (e.g., cat misclassified as dog).

Metric	Formula
Accuracy	$\frac{\sum TP}{\text{Total Samples}}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1 Score	$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$
Confusion Matrix	Raw counts of TP, FP, FN, etc.

Object Detection

What It Does

Object detection not only tells you what is in the image, but also where it is.

It gives:

- Class label (e.g., "dog")
- Bounding box (e.g., x=110, y=65, width=150, height=200)
- Confidence score (e.g., 0.89)

Architecture Types

There are 2 main types of object detectors:

TRANSFER LEARNING (RESEARCH)

1. Two-Stage Detectors

Example: Faster R-CNN

How it works:

1. Stage 1: Region Proposal Network (RPN) suggests candidate object areas (called Regions of Interest - ROIs).
2. Stage 2: Each ROI is classified and refined.

Pros:

- High accuracy
- Cons:
- Slower (not ideal for real-time)

---

2. Single-Stage Detectors

Examples: YOLO, SSD

How it works:

- Directly predicts bounding boxes and class probabilities from the image in one go.

Pros:

- Extremely fast (real-time)
  Cons:
- Slightly lower accuracy than two-stage (but newer versions are closing that gap)

YOLO Architecture (Simplified)

1. Divides image into an S × S grid
2. Each cell:
   - Predicts bounding boxes
   - Predicts objectness score
   - Predicts class probabilities
3. Combines predictions → outputs final list of objects

In YOLOv5 and v8:

- Everything is learned end-to-end.
- Uses anchor boxes and confidence thresholds.
- Non-Max Suppression (NMS) removes overlapping boxes.

TRANSFER LEARNING (RESEARCH)

	Description
Metric	
IoU	Measures overlap between predicted box & ground truth (0 to 1)
mAP	mean Average Precision — averaged over all classes and IoU thresholds
AP@[IoU]	AP at specific IoU threshold (e.g., 0.5, 0.75)
Precision/Recall	Used to analyze false positives and false negatives
FPS	Speed: How many frames can the model process per second
Term	Meaning
Precision	Of all boxes predicted, how many were correct? (/All Preds)
Recall	Of all actual objects, how many did we detect? (/All GT)
Confidence score	Model's certainty (0 to 1) that this prediction is correct

Eg:

Detected 3 objects:

- person (92%) at [x1, y1, x2, y2]
- dog (88%) at [x1, y1, x2, y2]
- bicycle (79%) at [x1, y1, x2, y2]

Semantic Segmentation

TRANSFER LEARNING (RESEARCH)

What is it?

Semantic segmentation is the process of classifying every pixel in an image into a predefined category.

Unlike object detection (which gives bounding boxes), semantic segmentation gives a pixel-level mask.

Example:

Input image:

A street scene with cars, people, buildings, road

Output:

- Pixels labeled as:
  - road = gray
  - car = blue
  - pedestrian = red
  - building = orange
  - sky = cyan

Every pixel is assigned a semantic class, but not a unique object identity. That's what instance segmentation does .

Model	Highlights
U-Net	Very popular in medical imaging; uses encoder-decoder with skip connections
DeepLabV3(+):	Uses atrous (dilated) convolutions + pyramid pooling
FCN (Fully Convolutional Network)	First deep learning approach to semantic segmentation
SegFormer	Transformer-based, efficient and accurate

Evaluation Metrics

Metric	Meaning
Pixel Accuracy	% of correctly classified pixels
IoU (Jaccard Index)	Intersection over Union (per class)

TRANSFER LEARNING (RESEARCH)

Metric	Meaning
mIoU (mean IoU)	Average IoU over all classes
Dice Coefficient	Like F1-score, used for imbalanced segmentation (especially medical)

Common Datasets

- PASCAL VOC: 20 categories (person, car, dog, etc.)
- Cityscapes: Urban street scenes (great for ADAS projects)
- ADE20K: 150 categories (objects + stuff)
- COCO-Stuff: Adds “stuff” categories to COCO
- Medical: BraTS, ISIC, etc.

Feature	Semantic Segmentation
Granularity	Pixel-level
Output	Mask (same size as input image)
Good for	Spatial understanding
Limitation	Can't separate multiple instances of the same class (e.g., 2 people overlap)

4. Instance Segmentation

What Is It?

Instance Segmentation = Object Detection + Semantic Segmentation

It not only classifies every pixel but also:

- Separates each object instance, even if they belong to the same class.

Semantic vs Instance Segmentation

Task	Detects "what"	Detects "where"	Separates objects
Semantic Segmentation	✅	✅ (pixel-level)	❌ (groups same-class pixels together)

TRANSFER LEARNING (RESEARCH)

Task	Detects "what"	Detects "where"	Separates objects
Instance Segmentation	✅	✅ (pixel-level)	✅ (individual objects)

Example:

- Two dogs in an image
  - Semantic segmentation: All dog pixels labeled "dog"
  - Instance segmentation: Dog #1 mask, Dog #2 mask

Popular Models

Model	Key Idea
Mask R-CNN	Extends Faster R-CNN by adding a third branch to predict segmentation masks
YOLACT / YOLACT++	Real-time, combines YOLO speed with masks
SOLO / SOLOv2	Segments objects directly without bounding boxes
Detectron2	Facebook's full framework for detection/segmentation tasks

---

How Mask R-CNN Works (Simplified)

1. Backbone (e.g., ResNet) extracts features from input image
2. RPN (Region Proposal Network) finds likely object regions
3. RoIAlign crops these regions and sends to:
   - Classifier
   - Bounding box regressor
   - Mask branch (predicts a binary mask for each instance)

Evaluation Metrics

Metric	Meaning
IoU (for masks)	Intersection over Union of predicted mask vs true mask
mAP@[IoU]	Average Precision for masks over various IoU thresholds
Dice Score / F1 for masks	Especially in medical or binary segmentation

TRANSFER LEARNING (RESEARCH)

Metric	Meaning
Per-instance Precision/Recall	Measures individual object mask accuracy

---

Datasets

- COCO (Common Objects in Context) — popular for object detection + instance masks
- LVIS — large vocabulary + instance segmentation
- Cityscapes — includes instance labels
- Kitti-MOTS — autonomous driving, multi-object tracking + segmentation

---

Applications

- Medical: segment tumors, cells, or organs per patient
- Autonomous driving: identify individual pedestrians, cars
- Retail: product instance detection
- Robotics: pick up specific objects
- Image editing: object cutout, manipulation

Summary

Feature	Instance Segmentation
Granularity	Pixel-level
Separates Instances?	Yes
Use Case	Multi-object, spatially detailed analysis
Models	Mask R-CNN, YOLACT, SOLOv2

5. Pose Estimation

What is Pose Estimation?

Pose estimation refers to the task of locating key points (joints) of a person (or object) in an image or video.
In humans, these keypoints could be:

- Eyes
- Shoulders

TRANSFER LEARNING (RESEARCH)

- Knees
- Ankles
- Hands, wrists, elbows

It’s not just about finding a “person” — it’s about understanding how they're positioned.

Real-world Example:

Take an image of a person walking:

- Output: Coordinates for all 17 joints (like COCO format)
- You can reconstruct their skeleton and even determine their action (walking, jumping, etc.)

MOST POPULAR MODELS FOR POSE ESTIMATION

Model	Description
OpenPose	First multi-person real-time pose estimator
HRNet	Maintains high-resolution features throughout the network
MediaPipe Pose	Google’s ultra-fast real-time pose estimation (good for mobile)
DeepPose	First to use deep learning for pose (by Google)
PoseNet	TensorFlow.js-compatible, lightweight pose estimation

How It Works

1. Input Image
2. CNN extracts feature maps
3. For each keypoint (e.g., elbow), predict a heatmap where it's most likely to be
4. Output final coordinates of all keypoints (max value in heatmap)

Evaluation Metrics

Metric	Description
PCK (Percentage of Correct Keypoints)	A keypoint is correct if it's within a radius from the ground truth point
OKS (Object Keypoint Similarity)	Like IoU, but for keypoints — accounts for scale and location
mAP for keypoints	Average precision across all joints and thresholds

TRANSFER LEARNING (RESEARCH)

6. Face Recognition / Verification

What is it?

This task involves using a person's face to:

- ✅ Recognize who they are (Face Identification)
- ✅ Verify if two faces belong to the same person (Face Verification)

It's different from face detection, which only finds the location of faces in an image.

Examples:

- Face ID on iPhones → Face verification
- Facebook auto-tagging → Face recognition
- Security systems → both

Two Modes:

Mode	Description
1-to-1 (Verification)	Is this person A? → Yes/No (e.g., unlocking phone)
1-to-N (Recognition)	Who is this person among many? (e.g., attendance)

Popular Models

Model	Notes
FaceNet	Learns embeddings; distance-based verification (Google)
Dlib	Lightweight C++ library with Python bindings
ArcFace	Uses angular margin loss; highly accurate
DeepFace	High-level Python API using multiple models underneath
VGGFace2	Model trained on large-scale celebrity dataset
InsightFace	State-of-the-art, optimized for production

How It Works:

1. Face is detected
2. Extract embedding vector (e.g., 128D or 512D)

3. For comparison:
    - If distance < threshold → same person
    - Else → different

Common distance metrics:

- Cosine similarity
- Euclidean distance

Evaluation Metrics

Metric	Description
Accuracy	Correct matches / total comparisons
ROC Curve / AUC	Visualizes true positive vs false positive rates
FAR (False Acceptance Rate)	% of wrong matches accepted
FRR (False Rejection Rate)	% of correct matches rejected
EER (Equal Error Rate)	Point where FAR = FRR (used in benchmarks)

Datasets

- LFW (Labelled Faces in the Wild)
- MS-Celeb-1M
- VGGFace2
- CASIA-WebFace
- FaceScrub

7. OCR – Optical Character Recognition

What is OCR?

OCR stands for Optical Character Recognition — it's the task of extracting text from images.

This includes:

- Scanned documents
- Handwritten notes
- Street signs in photos
- License plates
- Screenshots of code or articles

TRANSFER LEARNING (RESEARCH)

OCR Pipeline (Simplified)

1. Text Detection
    - Locate where text is in the image (bounding boxes)
2. Text Recognition
    - Convert the image inside each box into readable characters

---

Popular Models & Frameworks

Tool/Model	Purpose
Tesseract	Most common open-source OCR engine (by Google)
EAST	Efficient and Accurate Scene Text Detector
CRAFT	Character-Region Awareness for text detection
CRNN	Combines CNN + RNN + CTC for robust recognition
TrOCR	Transformer-based OCR (Microsoft)
EasyOCR	High-level Python wrapper over deep OCR stack
PaddleOCR	Very accurate; supports 80+ languages

Evaluation Metrics

Metric	Description
CER (Character Error Rate)	% of characters incorrectly recognized
WER (Word Error Rate)	% of words with mistakes
BLEU score (if comparing to reference text)	How close the output matches ground truth
Precision/Recall (for detection stage)	How accurately it finds text locations

8. Image Captioning

What is Image Captioning?

Image captioning is the task of generating a natural language description of an image. It blends computer vision (to understand the image) and natural language processing (to generate sentences).

Example:

TRANSFER LEARNING (RESEARCH)

Input:
A photo of a man riding a horse on a beach.

Output (Caption):

“A man is riding a horse along the shoreline.”

How it Works (Conceptually)

1. CNN encoder extracts features from the image (e.g., ResNet, EfficientNet)
2. RNN / Transformer decoder generates words one by one based on those features
3. The process is trained on pairs of (image, caption)

Popular Models

Model	Description
Show and Tell	CNN + LSTM model by Google (first deep learning-based image captioning model)
Show, Attend and Tell	Adds attention to focus on parts of image while generating each word
NIC (Neural Image Captioner)	Early encoder-decoder framework using InceptionNet + LSTM
BLIP / BLIP-2	Vision-language model with transformer decoder
ViT + GPT combos	Transformers on both vision and text sides (zero-shot capable)
CLIP + Decoder	Uses CLIP image embeddings and language decoders like GPT-2

Evaluation Metrics

Metric	What it Measures
BLEU	N-gram overlap with reference captions (precision-like)
ROUGE	Measures recall of overlapping units (more NLP focused)
CIDEr	Measures consensus with multiple reference captions (best for captions)
METEOR	Accounts for synonyms, stemming — better linguistic match
SPICE	Evaluates scene-graph level meaning

Example BLEU Calculation:

TRANSFER LEARNING (RESEARCH)

GT Caption: "A dog is running"
Predicted: "A dog is playing"
→ 2 out of 3 words match → BLEU score ≈ 0.66

Datasets

- MS-COCO (standard dataset with 5 captions per image)
- Flickr8k / Flickr30k
- Visual Genome
- Conceptual Captions (web-scaled dataset)

Real-World Use Cases

- Image accessibility for the visually impaired (screen readers)
- Automatic alt-text generation for the web
- AI-assisted photo tagging
- Visual question answering (VQA) building blocks
- News/media caption automation

🔎 9. Image Super-Resolution

What is Image Super-Resolution?

Image Super-Resolution (SR) is the task of enhancing the resolution of a low-quality image — essentially turning a blurry or pixelated image into a sharper, clearer version.

You input:

- A low-resolution (LR) image
  It outputs:
- A high-resolution (HR) version of the same image with improved detail

---

Real-world Example:

Input:
32×32 pixel face

Output:
128×128 or even 512×512 face with enhanced details

TRANSFER LEARNING (RESEARCH)

Types of SR

Type	Description
Single Image SR (SISR)	Enhance one image at a time
Video SR	Enhance resolution of frames in a video
Multi-image SR	Fuse multiple low-res views into one better image

---

Popular Models

Model	Highlights
SRCNN	First deep learning model for SR (simple and elegant)
SRGAN	Introduced perceptual + adversarial loss for realism
ESRGAN	Enhanced SRGAN with better detail recovery
Real-ESRGAN	Trained on real-world image degradation — great for photos
SwinIR	Transformer-based, state-of-the-art SR quality
EDSR	Very deep CNN without batch normalization for performance

---

How It Works (Typical Flow):

1. Input: LR image (e.g., 64x64)
2. Upsampling Layer: Bicubic or learned
3. Deep CNN / GAN layers: Restore lost features (e.g., textures)
4. Output: HR image (e.g., 256x256)

Evaluation Metrics

Metric	What it measures
PSNR (Peak Signal-to-Noise Ratio)	Higher = better pixel-level accuracy
SSIM (Structural Similarity Index)	Measures perceptual similarity (0–1)
LPIPS (Learned Perceptual Image Patch Similarity)	Learned metric aligned with human judgment (lower = better)
FID (Fréchet Inception Distance)**	If using a GAN-based model, FID helps measure realism

---

Use Cases

- Upscaling old photos (AI photo enhancers)

TRANSFER LEARNING (RESEARCH)

- Satellite imagery (sharpen terrain, roads, etc.)
- Video streaming (improve video quality at low bandwidth)
- Forensics (enhance blurry CCTV frames)
- Medical imaging (CT, MRI clarity improvement)

10. Image Generation

What is Image Generation?

Image generation is the task of creating completely new images using AI models — either:

- From random noise (like GANs)
- From text prompts (like DALL·E or Stable Diffusion)
- From other images (like style transfer or image-to-image translation)

Examples

- Generate fake faces: "a photo of a person who doesn't exist"
- Text-to-image: "a futuristic car driving on Mars"
- Image editing: remove background or colorize black-and-white photos

---

Major Techniques in Image Generation

Method	Description
GANs (Generative Adversarial Networks)	Learn to create realistic images by pitting two networks (Generator vs Discriminator)
Diffusion Models	Start with random noise → gradually “denoise” into a high-quality image
VQ-VAE (Vector Quantized VAE)	Discrete latent representation learning
Autoregressive Models	Predict next pixel/patch (e.g., PixelCNN)

TRANSFER LEARNING (RESEARCH)

Method	Description
Text-to-Image	Uses both vision and language models (CLIP + UNet)

Popular Models

Model	Description
StyleGAN2 / StyleGAN3	State-of-the-art GANs for photorealistic face generation
BigGAN	High-res class-conditional generation
CycleGAN	Translates images across domains (horse ↔ zebra)
Stable Diffusion	Text-to-image generation with stunning detail
DALL·E 2	OpenAI's model that generates images from text prompts
Midjourney	Proprietary, highly stylized text-to-image generation
DreamBooth	Fine-tunes a model on *you* (personalized generation)

How GANs Work (Simplified)

1. Generator (G) tries to make fake images
2. Discriminator (D) tries to tell if they're fake or real
3. They train together until the fake images are indistinguishable from real

---

Evaluation Metrics

Metric	Measures
FID (Fréchet Inception Distance)	Closeness of generated to real data (lower is better)
IS (Inception Score)	How diverse and high-quality the images are
LPIPS	Measures perceptual similarity (used for image-to-image tasks)
Human Evaluation	Sometimes the best option for creativity tasks

---

Applications

TRANSFER LEARNING (RESEARCH)

AI art and design (Midjourney, DALL·E)

Avatar & face generation

Photo enhancement & editing

- Fashion try-ons or virtual product mockups
- Data augmentation for training CV models
- AI-generated video frames (future of animation)

Summary

Feature	Image Generation
Input	Noise / Text / Image
Output	Fully generated image
Top Models	StyleGAN, Stable Diffusion, DALL·E
Metrics	FID, IS, LPIPS
Creativity	💯 Unmatched — truly generative AI

11. 3D Reconstruction / Depth Estimation

What is it?

This task focuses on understanding the 3D structure of a scene or object from 2D images.

- Depth Estimation: Predicts how far each pixel is from the camera
- 3D Reconstruction: Builds a full 3D model (point cloud, mesh, or volume) from one or more 2D images

---

Example:

Input:
A photo of a road

Output:
A grayscale depth map where brighter pixels = closer

TRANSFER LEARNING (RESEARCH)

Or:
A 3D mesh of the object or environment

Types of Depth Estimation

Type	Description
Monocular	Predict depth from a single image
Stereo	Use two camera views (like human vision)
Multi-view	Use multiple images from different angles
RGB-D	Combine color + depth sensor (like Kinect)

Popular Models

Model	Notes
MonoDepth / MonoDepth2	Monocular depth estimation from a single RGB image
MiDaS	Multi-scale, trained on many datasets, generalizes well
DPT (Dense Prediction Transformer)	Transformer-based for high-quality depth maps
NeRF (Neural Radiance Fields)	Volumetric 3D rendering from multiple images
COLMAP	Traditional multi-view 3D reconstruction (SfM/SLAM)

Evaluation Metrics

Metric	Description
RMSE (Root Mean Square Error)	Distance between predicted & true depth (lower is better)
MAE (Mean Absolute Error)	Average difference in depth values
Abs Rel Error	Average error relative to true depth
Threshold Accuracy (δ)	% of pixels where prediction is within factor (e.g., $\delta < 1.25$)

Real-World Applications

- Autonomous vehicles: depth sensing for driving & collision avoidance
- AR/VR: creating immersive environments

TRANSFER LEARNING (RESEARCH)

- Medical imaging: reconstructing 3D scans from 2D slices
- Architecture & mapping: building 3D models of spaces
- Robotics: environment perception for grasping or navigation

12. Video Action Recognition

What is it?

Video action recognition is the task of classifying the action taking place in a video clip or sequence of frames.

You're not just identifying *what* is in the scene — you're understanding *what is happening over time*.

Example:

- A 3-second video of a person jumping → Model predicts: "jumping"
- A sports video → Predicts: "throwing a basketball", "kicking", "swimming"

Key Difference from Image Classification:

You're working with space + time, not just pixels in a static image.

That means temporal patterns matter — like motion, velocity, and frame changes.

Popular Models

Model	Description
C3D (3D ConvNet)	Applies 3D convolutions over space & time
I3D (Inflated 3D ConvNet)	Inflates 2D kernels into 3D, using pretrained image models
SlowFast Networks	One stream captures slow features (semantics), the other fast motion
TimeSformer	Pure transformer model for video action recognition
VideoMAE	Masked autoencoder for self-supervised video learning
MoViNet	Optimized for mobile & real-time performance

How It Works

TRANSFER LEARNING (RESEARCH)

1. Sample video frames (e.g., 16 frames)
2. Extract spatiotemporal features using CNNs or Transformers
3. Use temporal pooling or attention
4. Predict an action label (e.g., “running”)

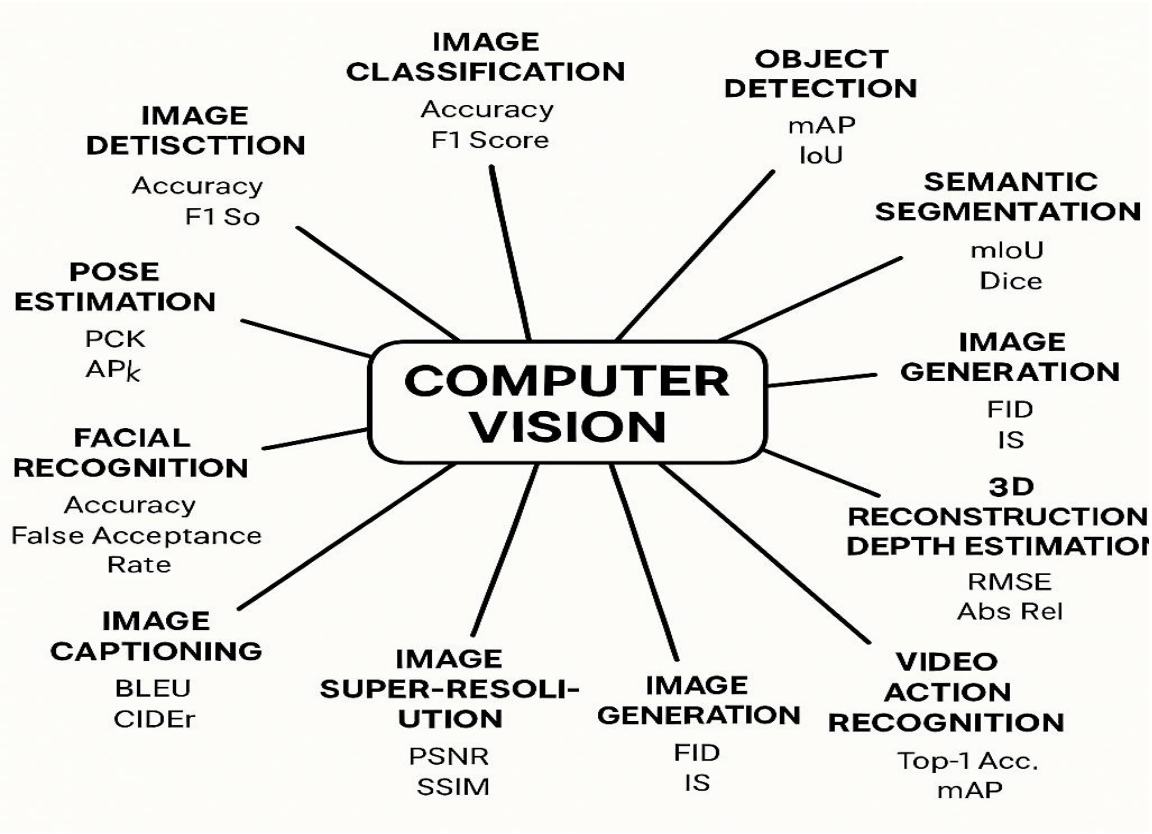
Evaluation Metrics

Metric	Description
Top-1 Accuracy	% of videos where the top prediction is correct
Top-5 Accuracy	% where correct label is in top 5 predictions
mAP (mean Average Precision)	Useful in multi-label settings
Precision / Recall	For per-action performance breakdown
Confusion Matrix	Shows what actions get confused with others

Datasets

Dataset	Description
UCF-101	101 action classes (sports, human activities)
Kinetics-400/600/700	Large-scale YouTube clips labeled with actions
HMDB-51	51 actions from movies and public video clips
Something-Something	Actions that require temporal understanding (e.g., “putting object on table”)
AVA	Annotated Video Actions with spatial + temporal localization

TRANSFER LEARNING (RESEARCH)



TRANSFER LEARNING (RESEARCH)

ASD vs NORM from Echocardiograms — Quadrant-Based Patch Learning (project)

Abstract

We made a classifier that classifies echocardiogram frames into ASD (diseased) vs NORM (normal) without chamber annotations. Each image is converted to grayscale, optionally center-cropped to suppress borders/text, and deterministically split into four quadrants corresponding to LA, RA, LV, RV. A ResNet-18 classifier is trained on the quadrants (patches) using: class-balanced sampling, class-weighted cross-entropy, strong augmentations, early stopping, and a two-phase fine-tuning schedule (head warmup → unfreeze layer4). At inference, per-patch predictions are stitched back to an image-level decision by majority vote. The system also produces stitched visualizations (original image with four boxes + per-quadrant predictions).

Problem Statement & Requirements

Goal: Predict ASD vs NORM from still echocardiogram images and visualize chamber-level reasoning.

Constraint: No chamber ground-truth boxes provided; IoU evaluation for RV/LV/RA/LA not feasible.

requirement:

Convert each image to grayscale.

Divide into four parts (LA, RA, LV, RV).

Train on patches.

Stitch back to a single prediction and show boxes.

Dataset Organization

data_orig/

train/{ASD,NORM}/*.png

validation/{ASD,NORM}/*.png

test/{ASD,NORM}/*.png

Provided counts (overall): ASD = 1500, NORM = 1332 (mild class imbalance).

TRANSFER LEARNING (RESEARCH)

Derived dataset is created as grayscale quadrants:

data_patches/

train/{ASD,NORM}/*_{LA|RA|LV|RV}.png

validation/{ASD,NORM}/*_{LA|RA|LV|RV}.png

test/{ASD,NORM}/*_{LA|RA|LV|RV}.png

patch_mapping.csv # traceability: full image ↔ its 4 patches

4) Method

4.1 Preprocessing

Grayscale conversion ("L"), then replicate to 3 channels for pretrained CNNs.

Quadrant split (fixed):

LA = top-left, RA = top-right, LV = bottom-left, RV = bottom-right.

4.2 Model

Backbone: ResNet-18 (ImageNet pretrained).

Head: Dropout(0.5) → Linear(num_features → 2) for {ASD, NORM}.

Input normalization: Normalize([0.5,0.5,0.5], [0.5,0.5,0.5]) (grayscale replicated).

4.3 Data Augmentation (train only)

RandomResizedCrop(224, scale=(0.5,1.0)), HorizontalFlip(0.5), Rotation(10°),

Affine(translate≈5%, scale 0.95–1.05, shear=5°), RandomErasing(p=0.5).

4.4 Handling Class Imbalance

WeightedRandomSampler with sample weights ∝ 1/class_count.Class

weighted CE (weights normalized to mean=1).

Label smoothing ε=0.05.

4.5 Optimization & Schedule

TRANSFER LEARNING (RESEARCH)

Two-phase fine-tuning

Warmup (epochs ≈ 3): freeze all; train head only with LR ≈ 3e-4.

Main phase: unfreeze layer4, LR ≈ 1e-4 (option to unfreeze layer3 later at 5e-5).

Optimizer: AdamW (weight decay 1e-4).

Scheduler: ReduceLROnPlateau on val accuracy (factor 0.5, patience=2).

Early stopping: patience=3 (best checkpoint saved whenever val-acc improves).

4.6 Inference & "Stitch-Back"

For each original image:

Split into 4 grayscale quadrants (LA/RA/LV/RV).

Predict per-patch probabilities $p_q \in \mathbb{R}^2$.

Aggregate to image prediction:

Majority vote: class with ≥3/4 votes (tie → average prob).

Average prob: p_final = (p_LA + p_RA + p_LV + p_RV)/4.

CSV output: final label + per-patch labels + all probabilities.

Visualization: draw four boxes on the original image and print each patch's prediction.

5) Experiments & Results

5.1 Training Behavior

Early naive runs (frozen backbone, weak aug) → severe ASD bias and overfitting (train ≈ 100%, val ≈ 50%).

With balancing, stronger aug, warmup + unfreeze + dropout + early stop → stabilized.

TRANSFER LEARNING (RESEARCH)

5.3 Outputs Produced

Model checkpoint: runs_quads_bal/cls/best_patch_cls.pth

Predictions CSV: runs_quads_bal/test_predictions_patch_agg.csv (or validation variants)

Visual overlays: runs_quads_bal/viz_quadrants/

Traceability: data_patches/patch_mapping.csv

6) Challenges Faced → Fixes Applied

No chamber GT boxes

Issue: Couldn't train/evaluate detection/IoU.

Fix: Use deterministic quadrants (LA/RA/LV/RV). allows patch training.

Everything predicted as ASD

Cause: Mild class imbalance + frozen backbone on grayscale domain.

Fixes: WeightedRandomSampler, class-weighted CE (+ label smoothing), unfreeze layer4, strong augmentation.

Overfitting (train ≈100%, val ≈40%)

Cause: Small dataset; clean quadrants too easy to memorize; border/text artifacts.

Fixes: Dropout 0.5, RandomErasing, center-crop option, early stopping, ReduceLROnPlateau.

Reproducibility / GPU setup

Fix: Scripts parameterized; best checkpoint auto-saved; CUDA build installed from PyTorch index; check_gpu.py sanity test.

7) Limitations

Quadrants are a heuristic; true chamber boundaries vary across views/patients.

Patch labels inherit global image label (weak supervision) → patch-level noise.

No chamber GT → no IoU and limited interpretability checks.

Borders/text/acquisition differences add nuisance variance.

Results are research-only (not diagnostic).

TRANSFER LEARNING (RESEARCH)

Experiments carried out to reach this point:

Experiment0 — Raw matrix .txt ingestion (early attempt)

Goal: Load echo frames stored as numeric matrices in .txt files and use them directly (or convert to images).

What we did:

Wrote a quick loader to visualize and optionally convert .txt → PNG:

Checked orientation (sometimes needed np.flipud/np.fliplr) and clipping for outliers.

Issues we hit:

Inconsistent formats: variable shapes (e.g., 256×256 vs 300×400), irregular delimiters, occasional headers/NaNs.

Dynamic range mismatch: some matrices were already scaled [0,1], others raw intensity → inconsistent contrast unless re-normalized per file.

Labeling friction: reliable labels came from folder names in the PNG dataset; .txt files often lacked clear ASD/NORM mapping.

Repro/time cost: custom readers increased complexity; batch I/O slower than using the curated PNGs

Experiment1 — Naïve quadrant pipeline (first attempt)

Goal: follow the mentor spec exactly (grayscale + 4 parts + stitch).

Setup

Preprocess: grayscale, split each image into LA/RA/LV/RV (equal 4 quadrants).

Model: ResNet-18; too much freezing (head only or minimal unfreeze).

Loss/imbalance: no balancing at this stage.

Augmentation: light (Resize/Flip/Small rotation).

Aggregation (“stitch-back”): average of per-patch softmax.

Observation

TRANSFER LEARNING (RESEARCH)

Failure mode: almost everything predicted ASD, including NORM—classic class-prior bias.

Evidence: predictions CSV dominated by ASD labels.

Takeaway

Quadrants work mechanically (boxes & CSV OK), but we need imbalance fixes + better fine-tuning.

Experiment2 — Add balancing + unfreeze last block (still overfits)

Goal: remove ASD bias and adapt features to grayscale.

Changes vs Experiment1

WeightedRandomSampler (oversamples NORM) + class-weighted CE (+ label smoothing 0.05).

Unfreeze layer4 (last ResNet block) so features can adapt.

Augmentation: moderate.

Early stopping + ReduceLROnPlateau enabled.

→ Overfitting persists.

Takeaway

Balancing fixed the “all-ASD” pathology, but with small/noisy patches the model still memorizes. We need stronger regularization and a gentler schedule.

Experiment3 — Robust trainer: stronger regularization + staged fine-tune (final)

Goal: improve generalization; stabilize training on small, noisy patches.

Changes vs E2

Grayscale-friendly normalization: mean/std = 0.5, 0.5, 0.5.

Stronger augmentation: RandomResizedCrop(224, scale=0.5–1.0), ±10° rotation, small affine, RandomErasing(0.5).

Higher dropout in head: 0.5 (was 0.2/none).

Two-phase schedule:

TRANSFER LEARNING (RESEARCH)

Warmup (3 epochs): freeze backbone; train head only (LR≈3e-4).

Fine-tune: unfreeze layer4, lower LR (≈1e-4), keep early stopping + LR scheduler.

Aggregation: switched default to majority vote (tie-break by avg prob); weighted option available (LV/RV heavier) but not needed for final.

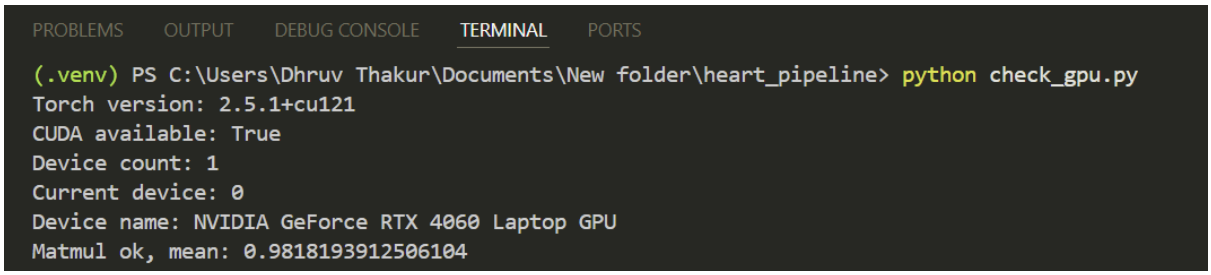
Result (validation, image-level)

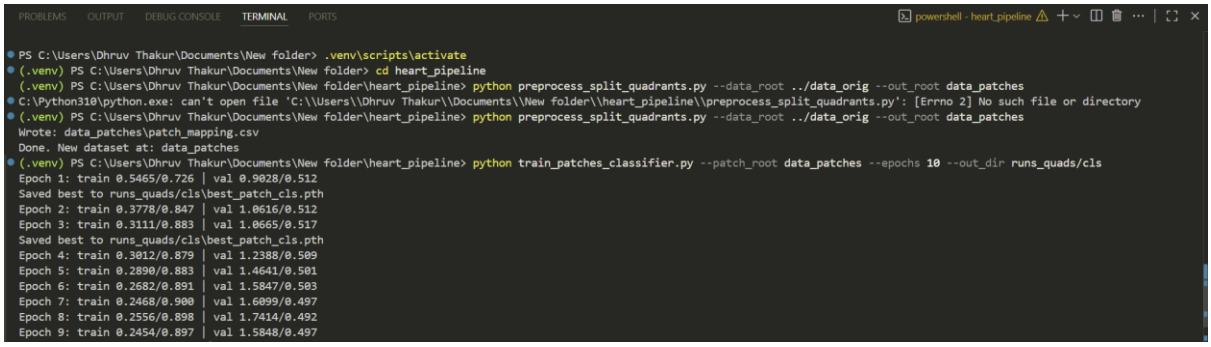
Training remained stable; early stopping saved the best checkpoint automatically.

Takeaway

The mix of balancing + strong aug + dropout + warmup→unfreeze + center-crop

Some screenshots of the outputs:





TRANSFER LEARNING (RESEARCH)

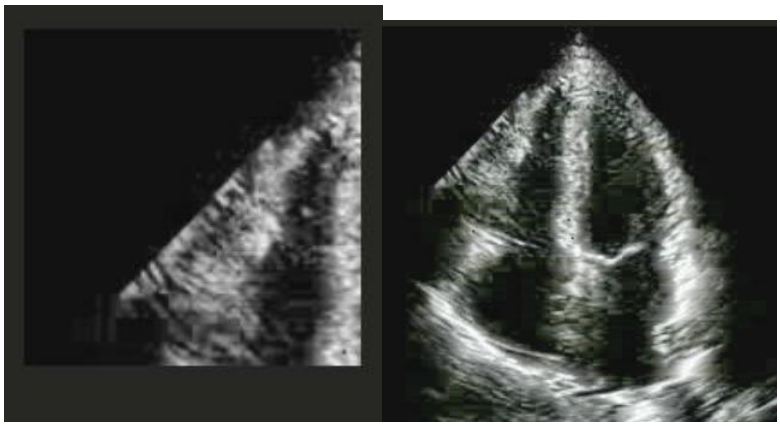
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    powershell - heart_pipeline

  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 1620, in __init__
    self._engine = self._make_engine(f, self.engine)
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 1880, in _make_engine
    self.handles = get_handle(
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\common.py", line 873, in get_handle
    handle = open(
FileNotFoundError: [Errno 2] No such file or directory: 'runs_quads/test_predictions_patch_agg.csv'
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> dir runs_quads


    Directory: C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline\runs_quads


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        31-08-2025     11:32                cls
d-----        31-08-2025     11:37                viz_quadrants
-a----        31-08-2025     11:37          59934 test_predictions_patch_agg.cs


(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> ren "runs_quads\test_predictions_patch_agg.cs" "test_predictions_patch_agg.csv"
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> python visualize_patch_boxes.py --data_root ../data_orig --pred_csv runs_quads/test_predictions_patch_agg.csv --
out_dir runs_quads/viz_quadrants
Saved visualizations to runs_quads/viz_quadrants
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline>

150%  224x224  67.74KB  Go Live
```

```
ls
Epoch 1: train 0.5465/0.726 | val 0.9028/0.512
Saved best to runs_quads/cls\best_patch_cls.pth
Epoch 2: train 0.3778/0.847 | val 1.0616/0.512
Epoch 3: train 0.3111/0.883 | val 1.0665/0.517
Saved best to runs_quads/cls\best_patch_cls.pth
Epoch 4: train 0.3012/0.879 | val 1.2388/0.509
Epoch 5: train 0.2890/0.883 | val 1.4641/0.501
Epoch 6: train 0.2682/0.891 | val 1.5847/0.503
Epoch 7: train 0.2468/0.900 | val 1.6099/0.497
Epoch 8: train 0.2556/0.898 | val 1.7414/0.492
Epoch 9: train 0.2454/0.897 | val 1.5848/0.497
Epoch 10: train 0.2374/0.907 | val 1.7643/0.500
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> python predict_fullimage_from_patches.py --data_root ../data_orig --weights runs_quads/cls/best_patch_cls.pth --
out_csv runs_quads/test_predictions_patch_agg.cs
C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline\predict_fullimage_from_patches.py:36: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defau
lt value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.
com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the fun
ctions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.
serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on Git
Hub for any issues related to this experimental feature.
  ckpt = torch.load(args.weights, map_location="cpu")
Wrote: runs_quads\test_predictions_patch_agg.cs
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> python visualize_patch_boxes.py --data_root ../data_orig --pred_csv runs_quads/test_predictions_patch_agg.csv --
out_dir runs_quads/viz_quadrants
Traceback (most recent call last):
  File "C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline\visualize_patch_boxes.py", line 15, in main
    df = pd.read_csv(args.pred_csv)
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 1026, in read_csv
    return _read(filepath_or_buffer, kwds)
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 620, in _read
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 1620, in __init__
    self._engine = self._make_engine(f, self.engine)
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\parsers\readers.py", line 1880, in _make_engine
    self.handles = get_handle(
  File "C:\Users\Dhruv Thakur\Documents\New folder\.venv\lib\site-packages\pandas\io\common.py", line 873, in get_handle
    handle = open(
FileNotFoundError: [Errno 2] No such file or directory: 'runs_quads/test_predictions_patch_agg.csv'
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> dir runs_quads


    Directory: C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline\runs_quads


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        31-08-2025     11:32                cls
d-----        31-08-2025     11:37                viz_quadrants
-a----        31-08-2025     11:37          59934 test_predictions_patch_agg.cs
```

TRANSFER LEARNING (RESEARCH)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    powershell - heart_pipeline

C:\Python310\python.exe: can't open file 'C:\\Users\\Dhruv Thakur\\Documents\\New folder\\heart_pipeline\\train_patches_classifier_balanced.py': [Errno 2] No such file or directory
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> cd
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> python train_patches_classifier_balanced.py --patch_root data_patches --epochs 20 --patience 3 --out_dir runs_qu
ads_bal/cls
Class counts (train): {'ASD': 1500, 'NORM': 1332}
Epoch 1: train 0.2292/0.944 | val 1.6325/0.466
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth (val_acc=0.4656)
Epoch 2: train 0.1500/0.995 | val 1.5143/0.479
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth (val_acc=0.4788)
Epoch 3: train 0.1465/0.996 | val 1.5933/0.466
No improvement for 1/3 epoch(s).
Epoch 4: train 0.1409/0.999 | val 1.5210/0.517
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth (val_acc=0.5172)
Epoch 5: train 0.1389/0.999 | val 1.3928/0.485
No improvement for 1/3 epoch(s).
Epoch 6: train 0.1373/0.998 | val 1.0889/0.519
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth (val_acc=0.5185)
Epoch 7: train 0.1328/0.999 | val 1.3091/0.483
No improvement for 1/3 epoch(s).
Epoch 8: train 0.1313/1.000 | val 1.0955/0.515
No improvement for 2/3 epoch(s).
Epoch 9: train 0.1344/0.999 | val 1.3244/0.504
No improvement for 3/3 epoch(s).
■ Early stopping: validation accuracy plateaued.
(.venv) PS C:\Users\Dhruv Thakur\Documents\New folder\heart_pipeline> python train_patches_classifier_balanced.py --patch_root data_patches --epochs 20 --warmup 3 --patience 3 --batc
h_size 32 --num_workers 0 --out_dir runs_quads_bal/cls
Class counts (train): {'ASD': 1500, 'NORM': 1332}
Epoch 01 [warmup-tr] loss=0.7583 acc=0.527 | lr 3.00e-04
Epoch 01 [warmup-va] loss=0.6836 acc=0.577
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth
Epoch 02 [warmup-tr] loss=0.6764 acc=0.618 | lr 3.00e-04
Epoch 02 [warmup-va] loss=0.6888 acc=0.597
 ✅ Saved best to runs_quads_bal/cls\best_patch_cls.pth
Epoch 03 [warmup-tr] loss=0.6254 acc=0.664 | lr 3.00e-04
Epoch 03 [warmup-va] loss=0.7899 acc=0.517
No improvement for 1/3 epoch(s).
Epoch 04 [train] loss=0.3408 acc=0.881 | lr 1.00e-04
Epoch 04 [valid] loss=1.4819 acc=0.466
No improvement for 1/3 epoch(s).
Epoch 05 [train] loss=0.2267 acc=0.954 | lr 1.00e-04
Epoch 05 [valid] loss=1.5105 acc=0.487
No improvement for 2/3 epoch(s).
Epoch 06 [train] loss=0.2050 acc=0.964 | lr 1.00e-04
Epoch 06 [valid] loss=1.0279 acc=0.548
No improvement for 3/3 epoch(s).
h/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could
be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_
safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues
 related to this experimental feature.
                                                                                                  Ln 134, Col 11  Spaces: 4  UTF-8  CRLF  {} Python  3.10.11 (.venv)  Go Live  Prettier
```

Preprocessing: ie splitting image into 4 quadrants

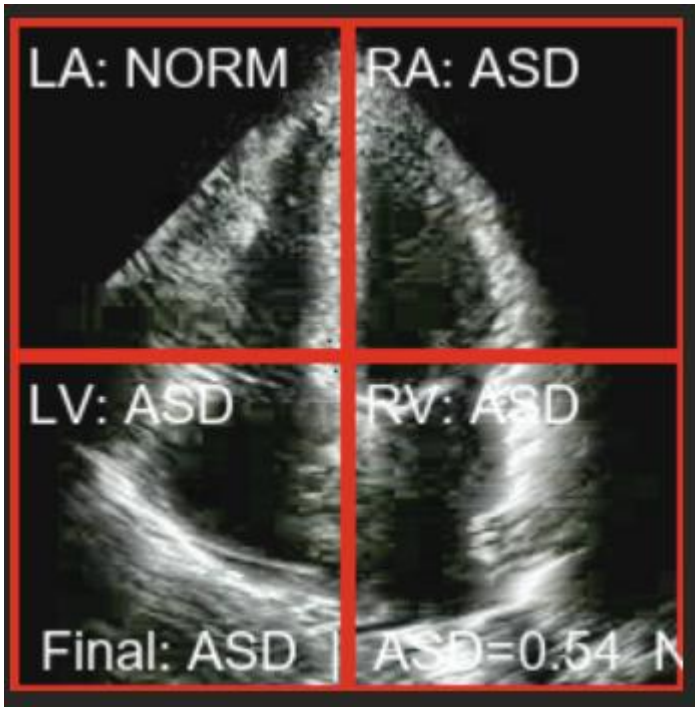


Left picture-> preprocessed image

Right picture->original image

Stiched back image after splitting

TRANSFER LEARNING (RESEARCH)



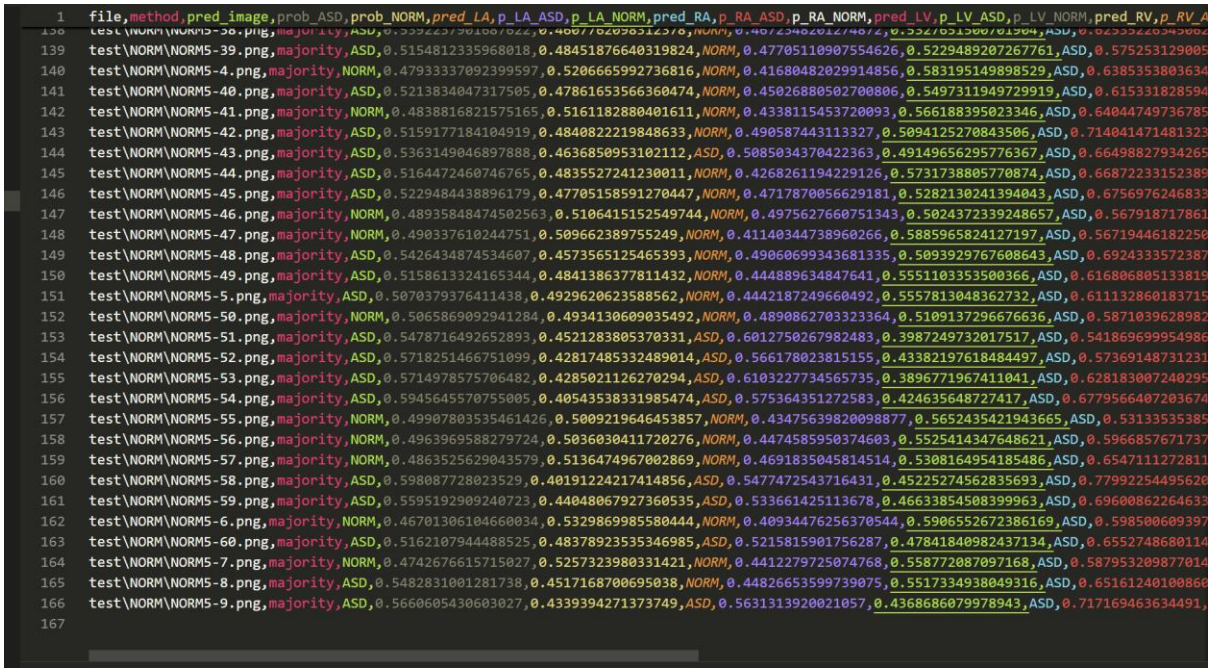
Predictions

heart_pipeline > runs_quads_bal > test_predictions_patch_agg.csv > data

```
file,method,pred_image,prob_ASD,prob_NORM,pred_LA,p_LA_ASD,p_LA_NORM,pred_RA,p_RA_ASD,p_RA_NORM,pred_LV,p_LV_ASD,p_LV_NORM,pred_RV,p_RV_A
test\ASD\ASD3-0.png,majority,ASD,0.5449343919754028,0.45506566762924194,NORM,0.41185408830642 7,0.5881459712982178,ASD,0.6996446847915649,
test\ASD\ASD3-1.png,majority,ASD,0.5474687814712524,0.45253121852874756,ASD,0.548130214214325,0.45186981558799744,NORM,0.4870705306529999
test\ASD\ASD3-10.png,majority,ASD,0.5507560968399048,0.4492439329624176,ASD,0.5064854621887207,0.4935145378112793,ASD,0.5754732489585876,
test\ASD\ASD3-100.png,majority,ASD,0.5787217020988464,0.42127829790115356,NORM,0.4579388201236725,0.5420611500740051,ASD,0.59075325727462
test\ASD\ASD3-101.png,majority,ASD,0.6236457228660583,0.3763542175292969,ASD,0.5452499389648438,0.45475006103515625,NORM,0.48676618933677
test\ASD\ASD3-102.png,majority,ASD,0.5520799160003662,0.4479200541973114,NORM,0.44863665103912354,0.5513634085655212,ASD,0.61760723590850
test\ASD\ASD3-103.png,majority,ASD,0.5655374526977539,0.4344625473022461,NORM,0.49436452984809875,0.5056354403495789,ASD,0.57345604896545
test\ASD\ASD3-11.png,majority,ASD,0.5611735582351685,0.43882644176483154,NORM,0.4809487462043762,0.5190512537956238,ASD,0.618202090263366
test\ASD\ASD3-12.png,majority,ASD,0.5404857397079468,0.459514319896698,NORM,0.3941248059272766,0.6058751940727234,ASD,0.6423219442367554,
test\ASD\ASD3-13.png,majority,ASD,0.5199623107910156,0.4800376892089844,NORM,0.4616115987300873,0.5383883714675903,ASD,0.6471217274665833
test\ASD\ASD3-14.png,majority,ASD,0.5767161846160889,0.42328381538391113,ASD,0.519439160823822,0.480560839176178,ASD,0.5447021722793579,0
test\ASD\ASD3-15.png,majority,ASD,0.5447009801864624,0.4552990198135376,NORM,0.475169837474823,0.524830162525177,ASD,0.6270598769187927,0
test\ASD\ASD3-16.png,majority,ASD,0.5303023457527161,0.46969759464263916,NORM,0.4951355457305908,0.5048643946647644,ASD,0.5407976508140566
test\ASD\ASD3-17.png,majority,ASD,0.5797778964042664,0.4202221035957336 4,NORM,0.4138944447040558,0.5861055254936218,ASD,0.6429958939552308
test\ASD\ASD3-18.png,majority,ASD,0.5075623989105225,0.49243760108947754,NORM,0.3945290148258209,0.6054710149765015,ASD,0.5303803682327277
test\ASD\ASD3-19.png,majority,ASD,0.5824604630470276,0.41753950715065,NORM,0.40804487466812134,0.5919551253318787,ASD,0.7269542217254639,
test\ASD\ASD3-2.png,majority,ASD,0.5761998295783997,0.42380014061927795,ASD,0.5885358452796936,0.4114641547203064,ASD,0.5574647188186646,
test\ASD\ASD3-20.png,majority,ASD,0.5673136115074158,0.43268635869026184,ASD,0.5041982661323547,0.49580973386764526,ASD,0.568570375442504
test\ASD\ASD3-21.png,majority,ASD,0.5824414491653442,0.41755855083465576,                                    5440711975,0.6189249753952026,ASD,0.67797935009002
test\ASD\ASD3-22.png,majority,ASD,0.5494822263717651,0.45051777362823486,NORM,0.4791386425495147 7,0.5208613276481628,ASD,0.51580363512039
test\ASD\ASD3-23.png,majority,ASD,0.5867162942886353,0.41328370571136475,NORM,0.44151991605758667,0.5584800839424133,ASD,0.5315924286842 3
test\ASD\ASD3-24.png,majority,ASD,0.5631487965583801,0.43685114383697 51,NORM,0.44988083839416504,0.550119161605835,ASD,0.5818806290626526
test\ASD\ASD3-25.png,majority,ASD,0.5679906010627747,0.43200942873954773,NORM,0.3928124606609344 5,0.6071875691413879,ASD,0.59400838613510
test\ASD\ASD3-26.png,majority,ASD,0.5503302812576294,0.4496697783470154,NORM,0.4885355532169342,0.5114644169807434,ASD,0.52901691198349,0
test\ASD\ASD3-27.png,majority,ASD,0.5622774362564087,0.4377225637435913,ASD,0.5699712634086609,0.4300287663936615,ASD,0.5981489419937134,
test\ASD\ASD3-28.png,majority,ASD,0.5429137349128723,0.4570862948894501,NORM,0.4410383105278015,0.5589616298675537,ASD,0.5353485941886902
test\ASD\ASD3-29.png,majority,ASD,0.5190001130104065,0.4809998571872711,NORM,0.4061344563961029,0.5938655138015747,ASD,0.5259858965873718
test\ASD\ASD3-3.png,majority,ASD,0.5065022706985474,0.49349772930145264,NORM,0.4267958104610443,0.5732041597366333,ASD,0.5308207869529724
test\ASD\ASD3-30.png,majority,ASD,0.4837392568588257,0.5162607431411743,ASD,0.5237081050872803,0.4762918949127197,ASD,0.5864952206611633,
test\ASD\ASD3-31.png,majority,ASD,0.5100356340408325,0.4899643659591675,NORM,0.4337042570114136,0.5662957429885864,ASD,0.5421732664108276
test\ASD\ASD3-32.png,majority,ASD,0.5122647285461426,0.4877353012561798,NORM,0.4154840111732483,0.5845160484313965,NORM,0.4211164116859433
```

Col 6: pred_LA

TRANSFER LEARNING (RESEARCH)



```
file,method,pred_image,prob_ASD,prob_NORM,pred_LA,p_LA_ASD,p_LA_NORM,pred_RA,p_RA_ASD,p_RA_NORM,pred_LV,p_LV_ASD,p_LV_NORM,pred_RV,p_RV_A
test\NORM\NORM5-39.png,majority,ASD,0.5154812335968018,0.48451876640319824,NORM,0.4770511090755462 6,0.5229489207267761,ASD,0.575253129005
test\NORM\NORM5-4.png,majority,NORM,0.47933337092399597,0.5206665992736816,NORM,0.41680482029914856,0.583195149898529,ASD,0.6385353803634
test\NORM\NORM5-40.png,majority,ASD,0.5213834047317505,0.47861653566360474,NORM,0.4502688050270080 6,0.5497311949729919,ASD,0.615331828594
test\NORM\NORM5-41.png,majority,NORM,0.4838816821575165,0.5161182880401611,NORM,0.4338115453720093,0.566188395023346,ASD,0.64044749736785
test\NORM\NORM5-42.png,majority,ASD,0.5159177184104919,0.4840822219848633,NORM,0.490587443113327,0.5094125270843506,ASD,0.714041471481323
test\NORM\NORM5-43.png,majority,ASD,0.5363149046897888,0.4636850953102112,ASD,0.5085034370422363,0.49149656295776367,ASD,0.6649882793426 5
test\NORM\NORM5-44.png,majority,ASD,0.5164472460746765,0.4835527241230011,NORM,0.4268261194229126,0.5731738805770874,ASD,0.668722331523 89
test\NORM\NORM5-45.png,majority,ASD,0.5229484438896179,0.47705158591270447,NORM,0.4717870056629181,0.528213024139404 3,ASD,0.675697624683 3
test\NORM\NORM5-46.png,majority,NORM,0.4893584847450256 3,0.5106415152549744,NORM,0.4975627660751343,0.5024372339248657,ASD,0.567918717861
test\NORM\NORM5-47.png,majority,NORM,0.490337610244751,0.509662389755249,NORM,0.41140344738960266,0.5885965824127197,ASD,0.567194468225 0
test\NORM\NORM5-48.png,majority,ASD,0.5426434874534607,0.4573565125465393,NORM,0.4906099343681335,0.5093929767608643,ASD,0.692433357238 7
test\NORM\NORM5-49.png,majority,ASD,0.5158613324165344,0.4841386377811432,NORM,0.444889634847641,0.5551103353500366,ASD,0.616806805133819
test\NORM\NORM5-5.png,majority,ASD,0.5070379376411438,0.4929620623588562,NORM,0.4442187249660492,0.5557813048362732,ASD,0.611132860183715
test\NORM\NORM5-50.png,majority,NORM,0.5065869092941284,0.4934130609035492,NORM,0.4890862703323364,0.5109137296676636,ASD,0.587103962898 2
test\NORM\NORM5-51.png,majority,ASD,0.5478716492652893,0.4521283805370331,ASD,0.6012750267982483,0.3987249732017517,ASD,0.541869699954986
test\NORM\NORM5-52.png,majority,ASD,0.5718251466751099,0.4281748533248901 4,ASD,0.566178023815155,0.43382197618484497,ASD,0.573691483123 1
test\NORM\NORM5-53.png,majority,ASD,0.5714978575706482,0.4285021126270294,ASD,0.6103227734565735,0.3896771967411041,ASD,0.628183007240295
test\NORM\NORM5-54.png,majority,ASD,0.5945645570755005,0.4054353833198547 4,ASD,0.575364351272583,0.4246356487274 17,ASD,0.677956640720367 4
test\NORM\NORM5-55.png,majority,NORM,0.4990780353546142 6,0.5009219646453857,NORM,0.4347563982009887 7,0.5652435421943665,ASD,0.5313353538 5
test\NORM\NORM5-56.png,majority,NORM,0.4963969588279724,0.5036030411720276,NORM,0.4474585950374603,0.5525414347648621,ASD,0.596685767173 7
test\NORM\NORM5-57.png,majority,NORM,0.4863525629043579,0.5136474967002869,NORM,0.4691835045814514,0.5308164954185486,ASD,0.654711127281 1
test\NORM\NORM5-58.png,majority,ASD,0.598087728023529,0.4019122421741485 6,ASD,0.5477472543716431,0.4522527456283569 3,ASD,0.779922544956 20
test\NORM\NORM5-59.png,majority,ASD,0.5595192909240723,0.4404806792736053 5,ASD,0.533661425113678,0.4663385450839996 3,ASD,0.696008622646 33
test\NORM\NORM5-6.png,majority,NORM,0.4670130610466003 4,0.5329869985580444,NORM,0.4093447625637054 4,0.5906552672386169,ASD,0.598500609397
test\NORM\NORM5-60.png,majority,ASD,0.5162107944488525,0.4837892353534698 5,ASD,0.5215815901756287,0.4784184098243713 4,ASD,0.655274868011 4
test\NORM\NORM5-7.png,majority,NORM,0.4742676615715027,0.5257323980331421,NORM,0.4412279725074768,0.558772087097168,ASD,0.587953209877014
test\NORM\NORM5-8.png,majority,ASD,0.5482831001281738,0.4517168700695038,NORM,0.4482665359973907 5,0.5517334938049316,ASD,0.651612401008 60
test\NORM\NORM5-9.png,majority,ASD,0.5660605430603027,0.4339394271373749,ASD,0.5631313920021057,0.4368686079978943,ASD,0.717169463634491,
```

Final metrics:

Confusion Matrix (Image-Level)

	index	Pred_ASD	Pred_NORM	
1	GT_ASD	98	6	
2	GT_NORM	48	13	

Per-Class Report (Precision/Recall/F1)

	class	precision	recall	f1
1	ASD	0.6712328767123288	0.9423076923076923	0.784
2	NORM	0.6842105263157895	0.2131147540983606 4	0.325

Per-Patch Accuracies

	quadrant	patch_accuracy	
1	LA	0.4	
2	RA	0.5696969696969697	
3	LV	0.8606060606060606	
4	RV	0.6121212121212121	

TRANSFER LEARNING (RESEARCH)

Q)What we actually did to get better accuracy?

1) Grayscale → Grayscale-friendly pipeline

Change: Convert every image to grayscale and switch normalization to mean=std=0.5 (instead of ImageNet RGB stats).

Why: Echo frames are intrinsically gray; removing color channels reduces noise/parameters and makes features easier to learn. Matching normalization to grayscale keeps activations well-scaled → steadier training.

2) Deterministic quadrant split

(LA/RA/LV/RV)Change: Split each frame into four equal quadrants; train on the patches; later stitch back to an image label.

Why: We had no chamber GT boxes, so this gives "chamber-like" regions without annotation, multiplies training examples ×4, and satisfies the mentor's "show boxes" requirement. It also encourages the model to learn local patterns.

3) Balanced sampling (fixes ASD bias)

Change: WeightedRandomSampler with per-sample weights $\propto$ 1/class_count.

Why: Even mild imbalance (ASD>NORM) made the model skew ASD. Oversampling NORM evens exposure each epoch → stops the "everything → ASD" failure mode.

4) Class-weighted cross-entropy + label smoothing

Change: CE with normalized class weights; label smoothing $\varepsilon \approx 0.05$.

Why: Further counteracts imbalance and reduces over-confidence, which helps validation generalization.

5) Stronger augmentation (harder to memorize)

Change: RandomResizedCrop(224, scale 0.5–1.0), flips, ±10° rotation, mild affine jitter, RandomErasing(0.5).

Why: Early training hit 100% train acc / ~50% val (overfitting). Heavier aug forces invariances the model must learn, improving out-of-sample accuracy.

6) Center-crop before splitting (optional but important)

Change: Keep ~85–90% center area prior to the 4-way split.

Why: Removes black borders and on-screen text that turn into misleading patches. Cleaner patches → less noise → better val accuracy.

7) Two-phase fine-tuning (warmup → unfreeze)

Change:

TRANSFER LEARNING (RESEARCH)

Warmup: freeze backbone, train head only for ~3 epochs (LR≈3e-4).

Main: unfreeze layer4 (optionally later layer3) with lower LR (≈1e-4 or 5e-5).

Why: ImageNet RGB filters don't perfectly match echo texture. A gentle, staged unfreeze lets the network adapt without blowing up gradients—reduces both bias and overfit.

8**) Higher dropout in the head (0.5 → not 0.2)**

Change: Dropout(p=0.5) before the final linear layer.

Why: Extra regularization against memorizing small, repetitive patches.

9) Early stopping + ReduceLROnPlateau

Change: Stop if val-acc doesn't improve for 3 epochs; cut LR by 0.5 on plateaus; save best checkpoint on every improvement

Why: Prevents wasting epochs where the model only memorizes training patches; always keeps the best model you reached.

10) Better "stitch-back" aggregation

Change: Use majority vote (tie → average prob)

Why: Simple averaging can still reflect prior bias; majority voting (or LV/RV-weighted) is more robust and clinically sensible.

What moved the needle the most (in practice)Balanced sampler + class-weighted loss → stopped the "all ASD" bias.

Stronger aug + RandomErasing + higher dropout → reduced overfitting.

Warmup + selective unfreezing with smaller LR → adapted features to grayscale echo texture.

Center-crop → removed border/text noise from patches.

Majority/weighted stitch → more robust final decision than plain averaging.

Q)Why unfreeze layers?

QFreezing a layer = don't let its weights update during training.

Unfreezing a layer = allow its weights to update again.

Why do this?

In transfer learning you start from a pretrained model (e.g., ImageNet). Early layers already extract general edges/textures; later layers are more task-specific.

TRANSFER LEARNING (RESEARCH)

Phase 1 (freeze): Train only a small “head” on top → fast, stable, avoids wrecking good pretrained features.

Phase 2 (unfreeze): Gradually allow deeper layers (e.g., layer4 in ResNet18) to learn your domain (echocardiogram grayscale) with a smaller LR so they adapt without forgetting