Getting started with Machine learning (SUBSIM)

by Vinzent Ferfers (2026)

Check-List of leaning-Content to start

1. Read article *(helped a lot)*
    a. ☒ Train, Validation, Test Split Explained (with Ratios)
    b. ☒ What is Data Augmentation? The Ultimate Guide.

2. Yolo tutorials link:(YOLO How-To Guides | Ultralytics)
    a. ☒ CV Data Collection and Annotation Guide | Ultralytics
    b. ☒ Preprocessing Annotated CV Data | Ultralytics
    c. – video needed, next article too complex at first
    ☒ *All Machine Learning algorithms explained in 17 min*
    https://youtu.be/E0Hmnixke2g?si=FXgVUHfaLOgtFcHp
    ☒ Backpropagation, intuitively | Deep Learning Chapter 3 (also chapter 1+2 very good)
    https://youtu.be/Ilg3gGewQ5U?si=QrCgpJ9UbpVM2KNk
    ☒ Model Training Tips & Best Practices | Ultralytics
    d. ☒ Fine-Tune YOLO26 on a Custom Dataset | Ultralytics
    e. ☒ YOLO Performance Metrics | Ultralytics

3. Try out SUBSIM Notebooks
    a. ☒ Make an account + order resources (needs moderator confirmation) 1d
    b. ☒ Datalab EDITO
    c. ☒ Biigle account (needs moderator confirmation)

4. A few papers from SUBSIM lab
    a. ☒ https://zenodo.org/records/21983917
    b. ☒ https://onlinelibrary.wiley.com/doi/10.1002/ece3.72091
    c. ☒ https://zenodo.org/records/17823220

Computer Vision:

YOLO = You Only Look Once

Data Collection and Annotation Strategies:

1. **Define Goals: specific, adjust needs, prepare**
2. **Class setup**:
    - Starting with more specific classes can be very helpful *(be as specific as possible in the setup, merging later possible if needed)*
        - Coarse Class-Count: These are broader, more inclusive categories, such as "vehicle" and "non-vehicle."
        - Fine Class-Count: More categories with finer distinctions, such as "sedan," "SUV," "pickup truck,"
3. **Avoiding Bias** in Data Collection:

- Diverse Sources
- Balanced Representation (small, big, old, young, etc)
- Continuous Monitoring
- Bias Mitigation Techniques: Use methods like oversampling underrepresented classes, data augmentation, and fairness-aware algorithms.

4. **Annotation Guidelines**
   - Clarity and Detail: Make sure your instructions are clear. Use examples and illustrations to show what's expected.
   - Consistency: Keep your annotations uniform e.g., only count fully visible animals not half on the edge, to improve consistency
   - Reducing Bias: Stay neutral, objective and minimize personal biases
   - Efficiency: Work smarter, not harder. Use tools and workflows that automate repetitive tasks, e.g. Biigle, Ultralytics Platform
5. **Identifying Outliers**
   - Statistical Techniques: detect outliers in numerical features; box plots, histograms, or z-scores
   - Visual Techniques: spot anomalies in categorical features; plotting images, labels, or heat maps
   - Algorithmic Methods: clustering (e.g., K-means clustering, DBSCAN) and anomaly detection algorithms to identify outliers based on data distribution patterns
6. **Quality Control of Annotated Data**
   - Reviewing samples of annotated data
   - automated tools to spot common errors
   - Having another person double-check the annotations (inter-annotator agreement means that the guidelines are clear)
7. **Efficient Data Labeling Strategies**
   - Clear Annotation Guidelines
   - Regular Quality Checks
   - Use Pre-annotation Tools
   - Implement Active Learning: prioritizes labeling the most informative samples first, which can reduce the total number of annotations needed while maintaining model performance
   - Batch Processing: Group similar images together for annotation
8. **Help platforms**
   - GitHub Issues: Visit the YOLO26 GitHub repository and use the Issues tab to raise questions, report bugs, and suggest features.

Data Preprocessing Techniques:

1. **Resizing Images:**
   Bilinear Interpolation: Smooths pixel values by taking a weighted average of the four nearest pixels
   Nearest Neighbor: Copies the nearest pixel value without averaging
2. **Normalizing Pixel Values:**
   (Min-Max Scaling, Z-Score Normalization, or automatic YOLO)
3. **Splitting Dataset:**

   For Big Datasets 10.000 + images **Train, Validation, Test (eg. 70/20/10) Split**

   - Maintain class distribution

- Balance classes: For imbalanced datasets, consider oversampling the minority class or under-sampling the majority class
- Split the dataset **before** applying any augmentation or other preprocessing

For small data sets **Cross-Validation** (not good for deep learning, resources better spent on more data)

4. **Augmenting the Data**
   - Mosaic, MixUp, and CutMix
   - Flips
   - Geometric transforms: Rotate, shift, zoom, and warp images.
   - HSV color jitter: Vary hue, saturation, and brightness.

5. **Exploratory Data Analysis (EDA)**
   - Start basic metrics: mean, median, standard deviation, and range
   - pixel-intensity distributions
   - Visual EDA Techniques
     - **Histograms and box plots** (catch outliers)
     - **Bar charts** (balance of each class)
     - **Scatter plots**: (image features, annotations)
     - **Heatmaps**: pixel-intensity distributions or the spatial distribution of annotations across images.

Model Training: Best Practices and Tips

1. **Theory**: adjusting its internal parameters to minimize errors. During training, the model iteratively makes predictions, calculates errors, and updates its parameters (weights and biases) through a process called backpropagation

2. **Training on Large Datasets (Methods/tips)**
   a. **Run GPU at max -1 :** never run out of GPU ram → lower batch size →automation in script *[batch=-1]* (based on your device's capabilities)
   b. **Subset Training**: script *[fraction=0.1]*
   c. **Multi-scale Training:** improves model's ability to generalize by training on images of varying sizes, detect objects at different scales and distances → with e.g. *[imgsz=640]* and *[multi_scale=0.25]*
   d. **Caching:** reduces the time the GPU spends waiting for data
      i. *[cache=True]:* Stores dataset images in RAM, providing the fastest access speed but at the cost of increased memory usage.
      ii. *[cache='disk']:* Stores the images on disk, slower than RAM but faster than loading fresh data each time.
   e. **Mixed precision training**: uses both 16-bit (FP16) for speed and 32-bit (FP32) for precision, floating-point types. Automatic Mixed Precision (AMP) training possible. *[amp=True]*
   f. **Pretrained weights:** Transfer learning adapts pretrained models to new, related tasks. Fine-tuning a pretrained model involves starting with these weights and then continuing training on your specific dataset.
   g. **Learning Rate Schedulers:** can prevent the model from overshooting minima and improve stability.

h. **Distributed Training**: spreading the training workload across multiple GPUs

3. **EPOCHS:** an epoch refers to one complete pass through the entire training dataset.
    a. starting point is 300 epochs. If the model overfits early, you can reduce the number of epochs. If overfitting does not occur after 300 epochs, extend the training.
    b. **Early stopping:** save computational resources and prevent overfitting
    → setting the patience parameter: [*patience=5*] means training will stop if there's no improvement in validation metrics for 5 consecutive epochs.

4. **Core training model settings**
    a. Choosing Between Cloud and Local Training
    b. Selecting an Optimizer: optimizer is an algorithm that adjusts the weights of your neural network to minimize the loss function
        i. SGD (Stochastic Gradient Descent)
        ii. Adam (Adaptive Moment Estimation)
        iii. RMSProp (Root Mean Square Propagation)
        iv. MuSGD (Muon + SGD hybrid)
    c. Fine-tune optimizer parameters
        i. For stability, you might start with a moderate learning rate
        ii. momentum determines how much influence past updates have on current updates common value 0.9. It generally provides a good balance

Fine-Tune YOLO on a Custom Dataset

1. Fine-tuning adapts a pretrained model to recognize new classes by starting from learned weights rather than random initialization
2. Tips for Fine tuning:
    a. When equivalent classes use different names, **rename** the source checkpoint classes in memory before loading.
    b. Freezing Layers: for smaller data set, prevents specific layers from updating during training. This speeds up training and reduces overfitting [*freeze=[0, 3, 5]]*
        i. *For large data sets better to not freeze*
3. **Key Hyperparameters** for Fine-Tuning
    The parameters that matter most are:
    a. **epochs**: Fine-tuning converges faster than training from scratch. Start with a moderate value and use patience to stop early when validation metrics plateau.
    b. **patience**: The default of 100 is designed for long training runs. Reducing this to 10-20 avoids wasting time on runs that have already converged.
    c. **Warmup epochs**: The default warmup (3 epochs) gradually increases the learning rate from zero, which prevents large gradient updates from damaging pretrained features in early iterations. Keeping the default is recommended even for fine-tuning.
4. **Two-stage fine-tuning**
    a. first stage freezes the backbone, trains only the neck and head, adapt to the new classes without disrupting pretrained feature
    b. The second stage unfreezes all layers and trains the full model with a lower learning rate
5. **Common Mistakes: Model produces no predictions**

a. Insufficient training data:
b. Check dataset paths
c. Lower confidence threshold
d. Verify class count: ensure nc in data.yaml matches the actual number of classes in the label files.

6. **mAP (mean Average Precision)**
    a. measures accuracy and precision (1= perfect detection; 0 = no detection)
    b. **mAP plateaus early** low score
        i. **Add more data**: additional training data, diverse examples with varied angles, lighting, and backgrounds.
        ii. **Check class balance:** underrepresented classes will have low AP. Use cls_pw to apply inverse frequency class weighting (start with cls_pw=0.25 for moderate imbalance, increase to 1.0 for severe imbalance).
        iii. **Reduce augmentation**: for very small datasets, heavy augmentation can hurt more than it helps. Try mosaic=0.5 or mosaic=0.0.
        iv. **Increase resolution:** for datasets with small objects, try imgsz=1280 to preserve detail.
7. **catastrophic forgetting** - the model loses previously learned knowledge when fine-tuned exclusively on new data. mitigate this:
    a. **Merge datasets**: include examples of the original classes alongside the new classes during fine-tuning.
    b. **Freeze backbone and neck**
    c. **Train for fewer epochs** on new data exclusevly

Object detection Metrics

1. **Intersection over Union (IoU):** IoU is a measure that quantifies the overlap between a predicted bounding box and a ground truth bounding box. It plays a fundamental role in evaluating the accuracy of object localization.
2. **Average Precision (AP):** AP computes the area under the precision-recall curve, providing a single value that encapsulates the model's precision and recall performance.
3. **Mean Average Precision (mAP):** mAP extends the concept of AP by calculating the average AP values across multiple object classes. This is useful in multi-class object detection scenarios to provide a comprehensive evaluation of the model's performance.
4. **Precision and Recall:**
    a. Precision quantifies the proportion of true positives among all positive predictions, assessing the model's capability to avoid false positives.
    b. Recall calculates the proportion of true positives among all actual positives, measuring the model's ability to detect all instances of a class.
5. **F1 Score:** The F1 Score is the harmonic mean of precision and recall, providing a balanced assessment of a model's performance while considering both false positives and false negatives.
6. **Class-wise Metrics**
    a. **Class:** name of the object class, such as "person", "car", or "dog".
    b. **Images:** number of images in the validation set that contain the object class.

c. **Instances**: count of how many times the class appears across all images in the validation set.
d. **Box (P, R, mAP50, mAP50-95):** This metric provides insights into the model's performance in detecting objects:
    i. **P (Precision):** indicating how many detections were correct.
    ii. **R (Recall):** measure of ability to identify all instances of objects in the images (is it missing some).
    iii. **mAP50:** Mean average precision calculated at an intersection over union (IoU) threshold of 0.50. It's a measure of the model's accuracy considering only the "easy" detections.
    iv. **mAP50-95:** The average of the mean average precision calculated at varying IoU thresholds, ranging from 0.50 to 0.95. It gives a comprehensive view of the model's performance across different levels of detection difficulty.

7. **Speed Metrics**
    a. e.g. Frame per seconds (FPS)
8. **Visual Outputs:**
    a. **F1 Score Curve** *(BoxF1_curve.png)*
    b. **Precision-Recall Curve** *(BoxPR_curve.png)*
    c. **Precision Curve** *(BoxP_curve.png)*
    d. **Confusion Matrix** *(confusion_matrix.png)*
    e. **Normalized Confusion Matrix** *(confusion_matrix_normalized.png)*
    f. **Validation Batch Labels** *(val_batchX_labels.jpg)*
    g. **Validation Batch Predictions** *(val_batchX_pred.jpg)*