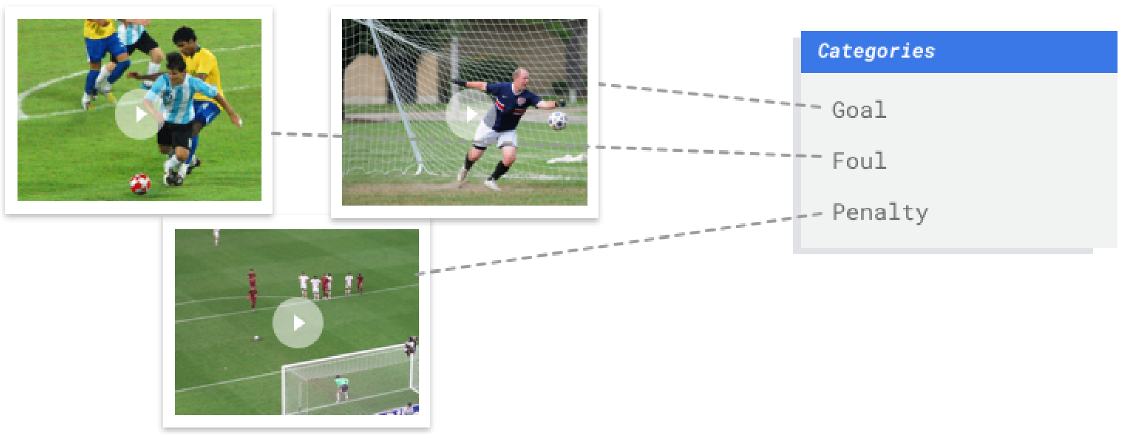
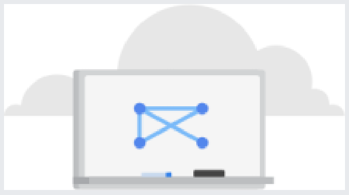
roduct is in a pre-release state and might change or have limited support. For more information, see the product launch stages lucts/#product-launch-stages).

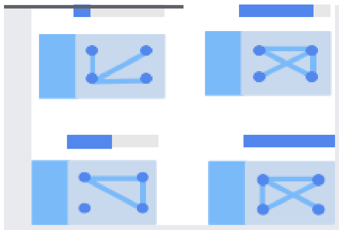
Imagine that you're a coach on a soccer team. You have a large video library of games that you'd like to use to study your team's strengths and weaknesses. It would be incredibly useful to compile actions like goals, fouls, and penalty kicks from many games into one video. But there are hundreds of hours of video to review and many actions to track. The work of watching each video and manually marking the segments to highlight each action is tedious and time-consuming. And you'll need to do this work for each season. Wouldn't it be easier to teach a computer to automatically identify and flag these actions whenever they appear in a video?





Classical programming requires a programmer to specify step-by-step instructions for a computer to follow. But consider the use case of identifying specific actions in soccer games. There's so much variation in color, angle, resolution, and lighting that it would require coding far too many rules to tell a machine how to make the correct decision. It's hard to imagine where you'd even begin. Fortunately, machine learning is well-positioned to solve this problem.

This guide walks you through how AutoML Video Intelligence Classification can solve this problem, its workflow, and the other kinds of problems it's designed to solve.

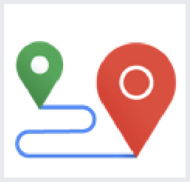


AutoML Video Intelligence Classification is a **supervised learning task**. This means that you train, test, and validate the machine learning model with example videos that are already labeled. With a trained model, you can input new videos and the model will output video segments with labels. A **label** is a predicted "answer" from the model. For instance, a trained model for the soccer use case would let you input new soccer videos and output video segments with labels describing action shots like "goal," "personal foul," and so on.

AutoML Video Intelligence Classification uses a standard machine learning workflow:

1. **Gather your data**: Determine the data you need for training and testing your model based on the outcome you want to achieve
2. **Prepare your data**: Make sure your data is properly formatted and labelled
3. **Train**: Set parameters and build your model
4. **Evaluate**: Review model metrics
5. **Deploy and predict**: Make your model available to use

But before you start gathering your data, you'll want to think about the problem you're trying to solve, which will inform your data requirements.



Start with your problem: What is the outcome you want to achieve? How many classes do you need to predict? A **class** is something you want your model to learn how to identify and is represented in the model output as a label (for example, a ball detection model will have two classes: "ball" and "no ball").

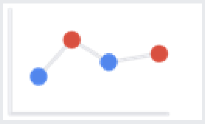
Depending on yours answers, AutoML Video Intelligence Classification will create the necessary model to solve your use case:

- A **binary classification** model predicts a binary outcome (one of two classes). Use this for yes or no questions, for example, identifying only goals in a soccer game ("Is this a goal or not a goal?"). In general, a binary classification problem requires less video data to train than other problems.
- A **multi-class classification** model predicts one class from two or more discrete classes. Use this to categorize video segments. For example, classifying segments of an Olympics video library to figure out which sport is being shown at any given time. The output provides video segments assigned a single label, like swimming or gymnastics.
- A **multi-label classification** model predicts one or more classes from many possible classes. Use this model to label multiple classes in a single video segment. This problem type often requires more training data because distinguishing between many classes is more complex.

The soccer example from earlier would require a multi-label classification mode, because the classes (actions like goals, personal fouls, etc) can occur simultaneously, which means a single video segment might require multiple labels.

Fairness is one of Google's responsible AI practices (https://ai.google/education/responsible-ai-practices). The goal of fairness is to understand and prevent unjust or prejudicial treatment to people based on race, income, sexual orientation, religion, gender, and other characteristics historically associated with discrimination and marginalization, when and where they manifest in algorithmic systems or algorithmically aided decision-making. As you read this guide, you'll see "Fair-aware" notes that talk more about how to create a fairer machine learning model. Learn more (/inclusive-ml/)

ware: Consider whether your use case can negatively impact individuals' economic or other life opportunities. If so, read more usive-ml/#assess-your-use-case) about assessing your use case for fairness considerations.



After you've established your use case, you'll need to gather the video data that will allow you to create the model you want. The data you gather for training informs the kind of problems you can solve. How many videos can you use? Do the videos contain enough examples for classes you want your model to predict? While gathering your video data, keep in mind the following considerations.

Generally, the more training videos in your dataset, the better your outcome. The number of recommended videos also scales with the complexity of the problem you're trying to solve. For example, you'll need less video data for a binary classification problem (predicting one class from two) than a multi-label problem (predicting one or more classes from many).

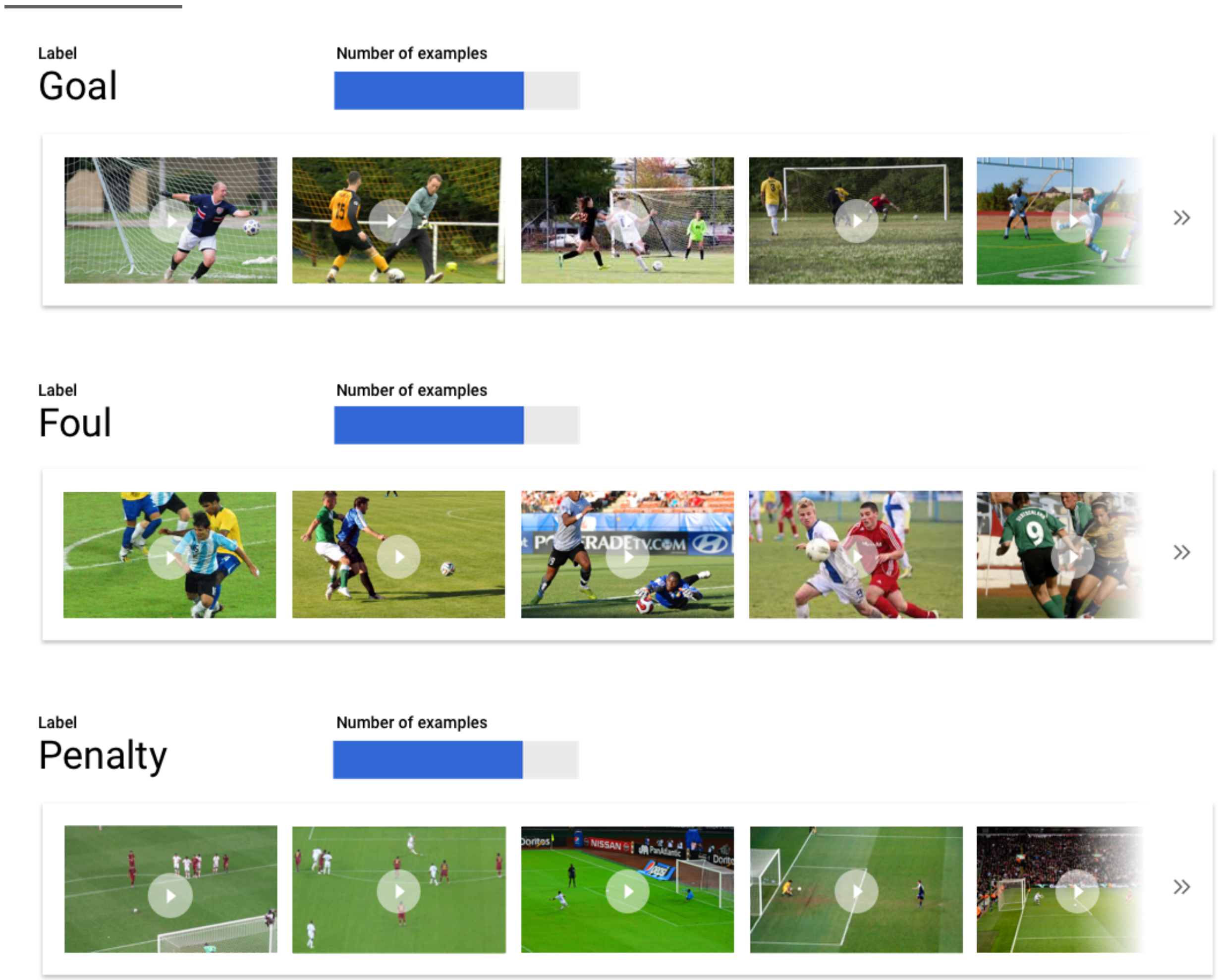
The complexity of *what* you're trying to classify can also determine how much video data you need. Consider the soccer use case, which is building a model to distinguish action shots. Compare that to a model distinguishing between hummingbird species. Consider the nuance and similarities in color, size, and shape: You'd need more training data in order for the model to learn how to accurately identify each species.

Use these rules as a baseline to understand your minimal video data needs:

- 200 video examples per class if you have few classes and they're distinctive
- 1,000+ video examples per class if you have more than 50 classes or if the classes are similar to each other

The amount of video data required may be more than you currently have. Consider obtaining more videos through a third-party provider. For instance, you could purchase or obtain more soccer videos if you don't have enough for your game action identifier model.

Try to provide a similar number of training examples for each class. Here's why: Imagine that 80% of your training dataset is soccer videos featuring goal shots, but only 20% of the videos depict personal fouls or penalty kicks. With such an unequal distribution of classes, your model is more likely to predict that a given action is a goal. It's similar to writing a multiple-choice test where 80% of the correct answers are "C": The savvy model will quickly figure out that "C" is a pretty good guess most of the time.



It may not be possible to source an equal number of videos for each class. High quality, unbiased examples may also be difficult for some classes. Try to follow a 1:10 ratio: if the largest class has 10,000 videos, the smallest should have at least 1,000 videos.

Your video data should capture the diversity of your problem space. The more diverse examples a model sees during training, the more readily it can generalize to new or less common examples. Think about the soccer action classification model: You'll want to include videos with a variety of camera angles, day and night times, and variety of player movements. Exposing the model to a variety of data will improve its model's ability to distinguish one action from another.

ware: While no training data is perfectly unbiased, you can improve your chances of building a more inclusive product if you identify potential ɛs of bias in your data and take steps to address them. Read more (/inclusive-ml/#data-guidelines).

Find training videos that are visually similar to the videos you plan to input into the model for prediction. For example, if all of your training videos are taken in the winter or in the evening, the lighting and color patterns in those environments will affect your model. If you then use that model to test videos taken in the summer or daylight, you may not receive accurate predictions.

Consider these additional factors: * Video resolution * Video frames per second * Camera angle * Background

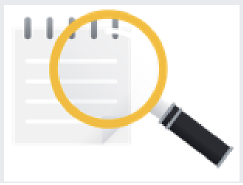
After you've gathered the videos you want to include in your dataset, you need to make sure the videos contain bounding boxes with labels so the model knows what to look for.

How does an AutoML Video Intelligence Classification model learn to identify patterns? That's what bounding boxes and labels do during training. Take the soccer example: each example video will need to contain bounding boxes around action shots. Those boxes also need labels like "goal," "personal foul," and "penalty kick" assigned to them. Otherwise the model won't know what to look for. Drawing boxes and assigning labels to your example videos can take time. If needed, consider using a labeling service (/data-labeling/docs) to outsource the work to others.

After your training video data is prepared, you're ready to create a machine learning model. Note that you can use the same dataset to make different machine learning models, even if they have different problem types.

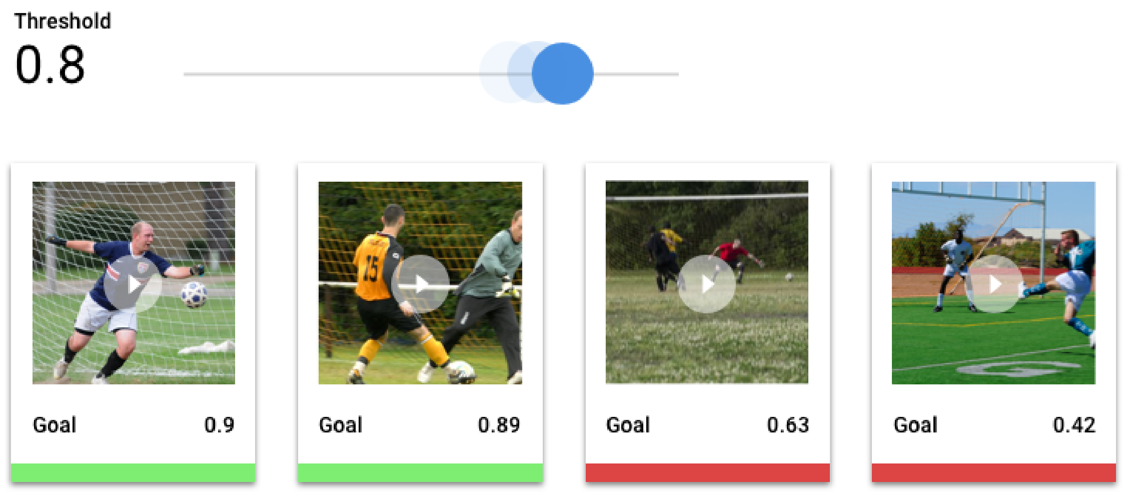
One of the benefits of AutoML Video Intelligence Classification is that the default parameters will guide you to a reliable machine learning model. But you may need to adjust parameters depending on your data quality and the outcome you're looking for. For example:

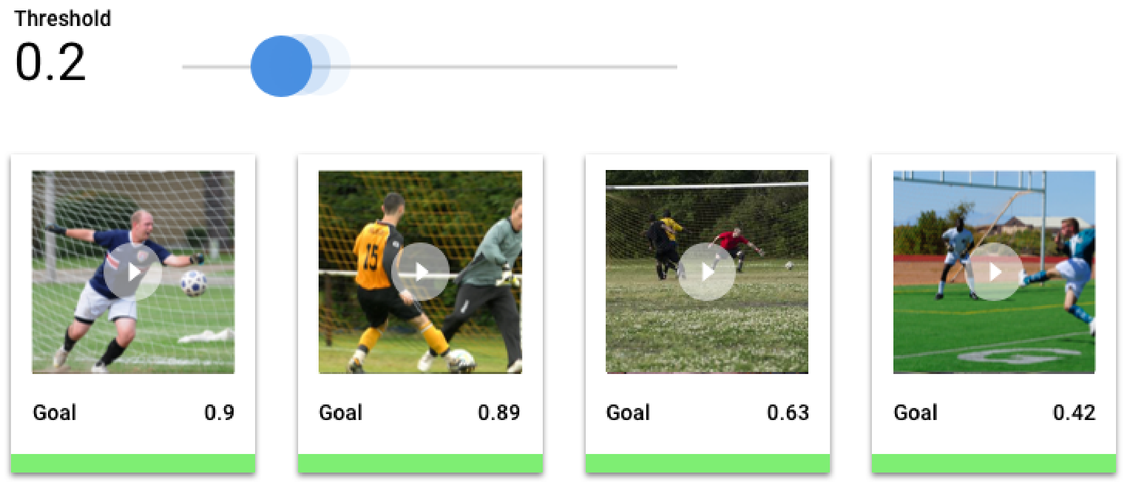
- Prediction type (the level of granularity which your videos are processed)
- Frame rate
- Resolution



After model training, you'll receive a summary of its performance. Model evaluation metrics are based on how the model performed against a slice of your dataset (the validation dataset). There are a couple key metrics and concepts to consider when determining whether your model is ready to be used in real data.

How does a machine learning model know when a soccer goal is really a goal? Each prediction is assigned a **confidence score** – a numeric assessment of the model's certainty that a given video segment contains a class. The **score threshold** is the number that determines when a given score is converted into a yes or no decision; that is, the value at which your model says "yes, this confidence number is high enough to conclude that this video segment contains a goal."

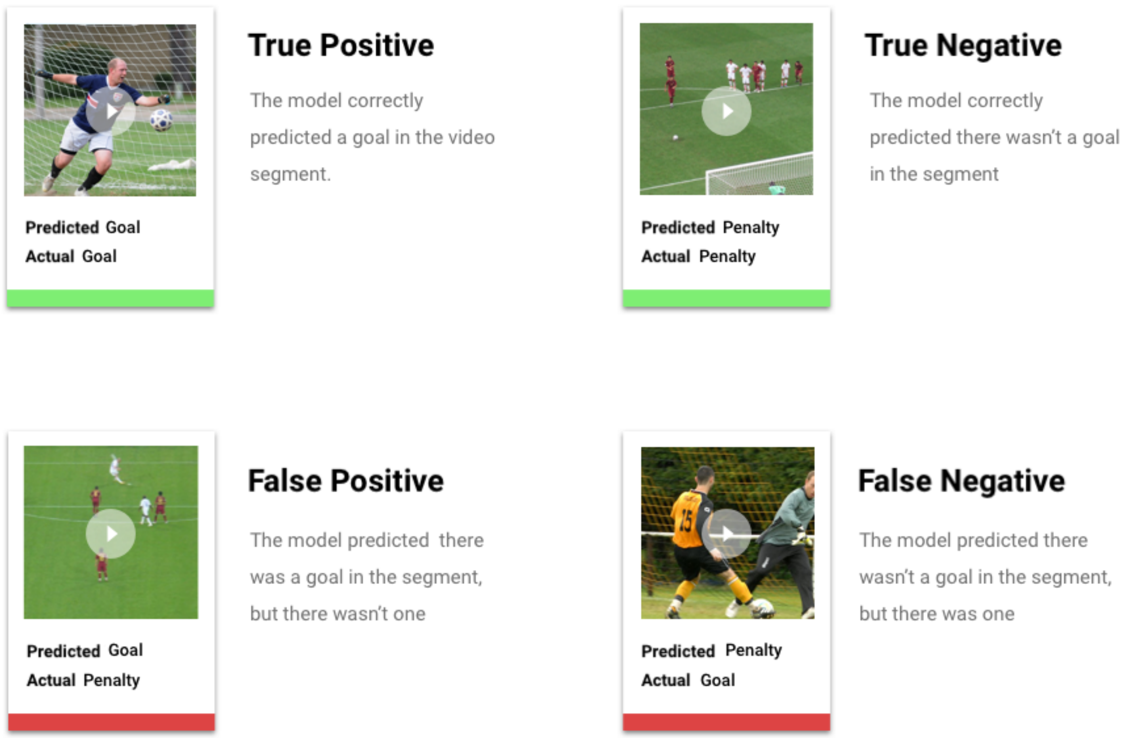




If your score threshold is low, your model will run the risk of mislabeling video segments. For that reason, the score threshold should be based on a given use case. Imagine a medical use case like cancer detection, where the consequences of mislabeling are higher than mislabeling sports videos. In cancer detection, a higher score threshold is appropriate.

After applying the score threshold, predictions made by your model will fall into one of four categories. To understand these categories, let's imagine that you built a model to detect whether a given segment contains a soccer goal (or not). In this example, a goal is the positive class (what the model is attempting to predict).

- **True positive**: The model correctly predicts the positive class. The model correctly predicted a goal in the video segment.
- **False positive**: The model incorrectly predicts the positive class. The model predicted a goal was in the segment, but there wasn't one.
- **True negative**: The model correctly predicts the negative class. The model correctly predicted there wasn't a goal in the segment.
- **False negative**: The model incorrectly predicts a negative class. The model predicted there wasn't a goal in the segment, but there was one.



True Positive
The model correctly predicted a goal in the video segment.
Predicted Goal
Actual Goal
True Negative
The model correctly predicted there wasn't a goal in the segment
Predicted Penalty
Actual Penalty
False Positive
The model predicted there was a goal in the segment, but there wasn't one
Predicted Goal
Actual Penalty
False Negative
The model predicted there wasn't a goal in the segment, but there was one
Predicted Penalty
Actual Goal

ware: Evaluating your model for fairness requires understanding the impact of different types of errors - false positives and false negatives - fc nt user demographics. Read more (/inclusive-ml/#evaluate).

Precision and recall metrics help you understand how well your model is capturing information and what it's leaving out. Learn more about precision and recall (https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall)

- **Precision** is the fraction of the positive predictions that were correct. Of all the predictions labeled "goal," what fraction actually contained a goal?

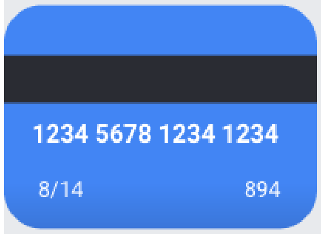
- **Recall** is the fraction of all positive predictions that were actually identified. Of all the soccer goals that could have been identified, what fraction were?

Depending on your use case, you may need to optimize for either precision or recall. Consider the following use cases.

Imagine you're building software that automatically detects sensitive information in a video and blurs it out. The ramifications of false outcomes may include:

- A false positive identifies something that doesn't need to be censored, but gets censored anyway. This might be annoying, but not detrimental.

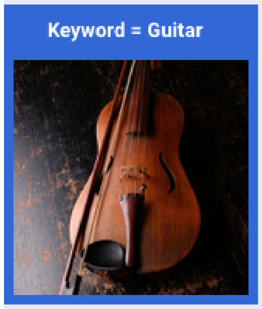
- A false negative fails to identify information that needs to be censored, like a credit card number. This would release private information and is the worst case scenario.



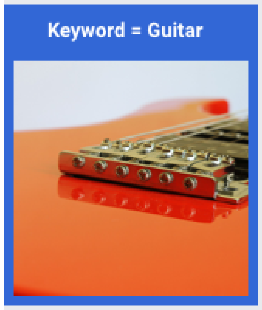
In this use case, it's critical to optimize for **recall** to ensure that the model finds all relevant cases. A model optimized for recall is more likely to label marginally relevant examples, but also likelier to label incorrect ones (blurring more than it needs to).

Let's say you want to create software that lets users search a library of videos based on a keyword. Let's consider the incorrect outcomes:

- A false positive returns an irrelevant video. Since your system is trying to provide only relevant videos, then your software isn't really doing what it's built to do.



- A false negative fails to return a relevant video. Since many keywords have hundreds of videos, this issue isn't a bad as returning an irrelevant video.



In this example, you'll want to optimize for **precision** to ensure your model delivers highly relevant, correct results. A high-precision model is likely to label only the most relevant examples, but may leave some out. Learn more about model evaluation metrics (/video-intelligence/automl/docs/evaluate)

ware: Think about your problem domain and its potential for unfairness and bias. Come up with cases that would adversely impact your users ose first. See an analysis of two use cases from an ML fairness lens (/inclusive-ml/#predict).

When you're satisfied with your model's performance, it's time to use your model. AutoML Video Intelligence Classification uses batch prediction, which lets you upload a CSV file with file paths to videos hosted on Cloud Storage. Your model will process each video and output predictions in another CSV file. Batch prediction is asynchronous, meaning that the model will process all of the prediction requests first before outputting the results.

ware: If you have a use case that warrants fairness considerations, read more about how to use your model in a manner that mitigates biases ;e outcomes. Read more (/inclusive-ml/#use-your-model).