UNSTRUCTURED DATA IS MORE WORTHLESS

Get Started Using Deep Learning for Business Users & Techies
Introduction

Do you have to manually review text or images?

“Content is King” after all. When you harness the power of unstructured data (for example, the data you thought wasn’t useful), you’re able to create more intelligent and streamlined workflows. Einstein Vision and Einstein Language allow Salesforce admins and developers to embed deep learning into their applications and create AI-powered apps, fast. Seriously. You’ll be a genius in no time.

For now, the main thing to remember is that deep learning technologies like image recognition and natural language processing help you:

- Simplify complex technologies so you can create more robust apps
- Build custom models or use pre-trained models for any use case
- Extend AI into any business process or CRM app using the same platform that you know and love

Ready?
Expert Level: Business User
First Things First

Terminology

Get started by understanding the terminology of data science and what to expect from the deep learning development life cycle.

**Data science** – a broad interdisciplinary field that uses automated, scientific methods to analyze large amounts of data and extract knowledge from that data.

**Artificial intelligence (AI)** – an umbrella term that refers to teaching computers to perform complex tasks. Robots who play chess and the personal assistant on your phone? Those are examples of AI.

**Machine learning** – a technique that helps a computer learn without being explicitly programmed with step-by-step instructions like you see in AI.

**Deep learning** – a subset of machine learning where you create and train a neural network to recognize patterns.

**Dataset** – contains the training data from which the model is created.

**Label (Class)** – references the output name you want your model to predict. For example, the food classifier, which is trained from a multi-label dataset, contains labels like chocolate cake, pasta, carrots, and so on.

**Training** – the process by which a model is created.

**Model** – a machine-learning construct used to solve a classification problem.

**Prediction** – the results that the model returns as to how closely the input matches data in the dataset.

**Model metrics** – a picture of model accuracy and how well the model will perform.

Understanding the Deep Learning Development Cycle

**Good Data is Key**

Data is the means by which a model is created. So your model is only as good as the data on which it’s based. After you define your use case, or the problem that you’re solving, you must determine if you have enough data. If you don’t have enough good-quality data, the first problem to solve is how to get the data required for your implementation.

For example, you want to create a model that analyzes phrases from chat transcripts to identify what a user is trying to accomplish. To create such a model, you need existing chat data. If you implemented chatbot functionality a month ago, you might not yet have enough data to build an accurate model. Whereas, if you have six months of transcripts from an active chatbot, then you might have enough data. It helps to start with a minimal viable dataset and then iterate and grow your dataset as you acquire more data.

“If you don’t have enough good-quality data, the first problem to solve is how to get the data required for your implementation.”
Gathering Data Takes Time

This point is a corollary to the previous principle. Even if you know that you have enough data available to create a model, chances are that data requires some processing.

If you’re working with text data, you might need to parse phrases out of a transcript. It’s likely that text data needs to be formatted in the way that the API expects it. If you’re working with image data for object detection, you need bounding box data for the items in each image.

At the start of your project, you must also define the categories or labels in which your data falls. In addition to gathering “correct” data examples to make predictions for the types of data you expect, you also want to gather “incorrect” data to create a negative label.

The Process is Iterative

The model creation process involves frequent loops of model creation and testing. The overall process looks like this:

1. Use initial data to create the dataset and model.
2. Test the model.
3. Refine the dataset based on the model test results.
4. Create a new test model, and compare to the previous models.
5. When you’re satisfied with the results, create a final production model from the most recent dataset.

You repeat steps 2–4 as often as necessary until you end up with an accurate model. By using Einstein Vision and Einstein Language, you can focus on the data and quickly get a prototype model up and running. You can use these APIs to get model metrics, test the accuracy, and quickly iterate. You can get started with the minimum viable dataset and iterate and grow your dataset and labels as you acquire more data.

Improvement is Ongoing

After a model is in production, you continue to find ways to improve it. For example, you can track when the model makes an inaccurate prediction. You then use the feedback API to add that data with the correct label to the dataset, and retrain the dataset to update the model.

You might find that certain data is consistently misclassified, perhaps data that you didn’t expect to be sent for classification. In this case, you might want to add a label and the data to the dataset and retrain it. Your business might change and require adding a new label and data to the dataset and model.

Expect that your models continue to change even after they’re in production. It’s important to expect these changes so that you can allocate resources and time to maintaining the model.

Now that you understand more about deep learning and the development cycle, it’s time to investigate how you can use Einstein Vision and Einstein Language to solve your own business challenges.
Expert Level: Business User
Making a Plan

Selecting the Best Use Case

Here’s a list of considerations when selecting a successful use case.

Select the Right Business Challenge

During the process of identifying a good use case, there are various factors to consider. First, find business challenges in your organization that fit the technologies available in Einstein Vision or Einstein Language. Look for tasks that can be automated and are a good match for these services. Be sure that the challenge you solve isn’t too big or complex. This way you can achieve success more quickly. And consider whether solving the challenge has value to your organization.

“Look for tasks that can be automated and are a good match for these services.”

Select a Use Case for Which You Have Data

Data is central to successful deep learning projects. When you identify a high-value business challenge, confirm that you have enough data to solve the challenge using deep learning. The quality of the data is also important. For Einstein Vision, do you have a variety of images of what you want to identify? For Einstein Language, do you have enough sentiment or intent text data for each label? If not, you can usually find datasets online which can give you a headstart.

Understand the Effort Required for Data Processing (Labeling or Formatting Data)

After you identify that you have enough high-quality data, be prepared for the time and effort required to process the data. Whether you use Einstein Vision or Einstein Language, the data must be sorted and labeled. For object detection, parts of images must be labeled with bounding box data. If text data is in long paragraphs, you need to process it so that individual phrases can be extracted and labeled.

Define the Business Success Metrics

When you implement deep learning, your goal isn’t to use AI; the goal is to solve a business problem. To ensure that your solution is working, define what success looks like from the business perspective. For example, if you’re implementing a machine learning project to route incoming support cases, define up front how many cases you want to route correctly and in what timeframe. If your solution correctly routes a case, identify the time and cost savings. Understand the impact that the solution has on your business.

Bring in the Right Expertise

To build and implement deep learning solutions requires people in your organization that understand AI and deep learning. You can use Einstein Vision and Einstein Language to quickly build solutions. But it’s still necessary to have people that understand the deep learning life cycle and technologies. Be sure that you have people that can frame the problem, collect the right data, label and process the data, understand the model, interpret the model metrics, and understand how to use feedback to improve the model. It’s also helpful to have someone who owns project management (assigns deliverables, owners, deadlines) to make sure the project is a success.
Rolling it Out

Define your rollout plan, and understand how to manage your development. Get familiar with how datasets, training, models, and predictions work.

Managing Development

Einstein Vision and Einstein Language include REST-based APIs to help you quickly build image classification or natural language processing into your apps. These APIs have a few key differences from other Salesforce APIs, like the REST API or the Chatter REST API.

- Although you can use Salesforce or Heroku to sign up for an Einstein Platform Services account, the API endpoints are outside of your Salesforce app. For example, the Salesforce REST API endpoint to get an account might look like:

  https://yourInstance.salesforce.com/services/data/v44.0/sobjects/Account/<accountId>

- The Einstein Vision API endpoint to get a dataset might look like:

  https://api.einstein.ai/v2/vision/datasets/<datasetId>

- Einstein Vision and Einstein Language use OAuth 2.0 JWT bearer token flow for authorization. This authorization method is different from the authorization methods you can use with the Salesforce REST APIs, which are OAuth 2.0 web server flow, user-agent flow, or username-password flow.

- Each Einstein API call requires a valid access token. You programmatically handle getting an access token by using your key or by generating a refresh token with your key to get an access token. When you use the Einstein Vision or Einstein Language APIs, calls to get a token are made to the endpoint:


- When you customize the Salesforce app, you can use a sandbox environment, which is typically a copy of your production org, to test your changes. Because the Einstein Vision and Einstein Language APIs are outside of Salesforce, you define your own testing and deployment environments. For example, you could have some test models that you create using an account for testing. Then when you decide one of those models is ready for production, you can create that dataset and model using your production account.

Working with Datasets and Models

A dataset contains the source image or text data. A model is created after a dataset is trained. The model is the construct that returns predictions.

Here are some key points about datasets and models:

- A dataset is the structure that contains your data, whether that data is image or text.

- The training process uses the dataset to create a model. You train a dataset multiple times to create multiple models. So a single dataset can create many models.

- The relationship between a dataset and a model is complete after the model is created. After a model is created, the model doesn’t reference the source dataset again unless you retrain the dataset.

- You use APIs to edit a dataset. There’s no visual way of editing a dataset. If you have new data that you want to include in a model, you call the API to create a new dataset, or you can call the API to add the new data to an existing dataset.

Data Collection Tips

- For Einstein Language, target 200–500 examples per dataset label. For Einstein Vision, target at least 1,000 images per label.

- Make sure that each dataset label has about the same number of images. For example, avoid a situation where you have 1,000 examples in one label and 400 in another in the same dataset.

- Use a wide variety of examples per label to improve the model accuracy, because images or text sent in for prediction likely vary. Make sure your examples are representative of the data you want the model to analyze.
Consider including a negative label, such as “Other.” When an image or text doesn’t match anything in the model, the model returns the value “Other.”

**Training and Retraining**

The training and retraining processes both create a model, but they are slightly different. There are different cases when you want to use train or retrain, and this content helps you understand when and why to use each method.

**Training** is the process by which a model is created from your data (a dataset). In the training process, algorithms are combined with your data to create the object that returns predictions.

Each time you call `/train` to train a dataset, it creates another model with a different model ID. Creating a separate model each time you train a dataset is helpful because you can compare them and then test and refine your models.

For example, you train dataset A, create a model, and find that the model doesn’t perform to the level you expect. So you add some data to the dataset and train that dataset to create a new model. You can still access the previous model to compare the accuracy and other metrics as you iterate through the process.

Here are some considerations to keep in mind when you train a dataset:

- When you call `/train`, it creates a new model with a new model ID.
- If you decide to use the new model, you must update the model ID in existing code or tools that reference the model ID.
- Creating multiple models enables you to compare models as you change your dataset.

**Retraining** works differently. When you call `/retrain` to train a dataset, it references the dataset used to create an existing model. This method overwrites the old model with the new model, but it retains the previous model ID.

For example, you train a dataset, create a model, and iterate through the process. You have an accurate model that you use in your apps. Due to changes in your business, you need to include more data in your model. You add the necessary data to your dataset and then call `/retrain` to refresh the existing model.

Here are some considerations to keep in mind when you retrain a dataset:

- Retraining overwrites the existing model, but maintains the model ID.
- If you retrain using bad or incorrect data, the model is corrupted. You can’t recover the older version.
- Because retraining keeps the original model ID, you don’t need to update references to the model ID. Your existing code and tools use the new model.
- You must explicitly retrain a dataset when you want to update model. Retraining doesn’t occur automatically.
- You can only retrain a dataset from which a model was created. For example, if you have dataset 1 and model A, you can add feedback data to dataset 1. You can retrain dataset 1 multiple times to update model A. However, you can’t create dataset 2 and retrain to create model A.
Training Data and Test Data

When you train or retrain a dataset, the training process sets aside some of the training data to test the model for accuracy.

**Training data** – Examples used by the training process to create the model.

**Test data** – Examples set aside by the training process to test the model's accuracy.

The amount of data used for training and testing varies depending on which API you use.

- **Einstein Vision** – 90% of the data is used to create the model, and 10% is used to test the model's accuracy.

- **Einstein Language** – 80% of the data is used to create the model, and 20% is used to test the model's accuracy.

You can change this ratio, also called the split, by using the `trainSplitRatio` parameter.

You specify the amount of training and test data, but the actual data that the training process holds out for testing is randomly selected. This means you can see differences in models and model metrics even when those models are created from the same dataset.
Continually Improving
(The gift that keeps giving!!)

Keep iterating on your rollout plan by optimizing your API calls and using feedback to improve your model.

Best Practices

**Einstein Vision**

- Target at least 200–1,000 examples per dataset label.
- Include a wide variety of images for each dataset label. If you have a label that contains images of a certain object, include images with these characteristics:
  - Color
  - Black and white
  - Blurred
  - With other objects the object might typically be seen with
  - With text and without text (if applicable)

- A wide variety of images makes the model more accurate. For example, if you have a dataset label called “buildings,” include images of many different building styles: skyscraper, gothic, modern, and so on.
- In a binary dataset, include images in the negative label that look similar to images in the positive label. For example, if your positive label is oranges, include grapefruits, tangerines, lemons, and other citrus fruits in your negative label.
- For a multi-label model, include images with objects that appear in different areas within the image. For example, if you have images that have the label “car,” incorporate images that have the car in different areas within the image.

**Einstein Language**

- A dataset can have up to 500 labels, but we recommend a maximum of 100 labels for better model accuracy.
- If you have a dataset that contains a lot of classes, increase the number of examples per label.
- We recommend that an Einstein Intent or Einstein Sentiment dataset contain a maximum of 100 labels. If you need more than 100 labels, consider hierarchical classification.
- We recommend 50-100 words for the length of the intent or sentiment string.
- During the training process, special text formatting, like emojis, words in all uppercase, and punctuation aren’t included. For example, if you add a text example containing a smiley emoji to a dataset, the emoji isn’t considered during training. Only the text is used.
- When you send in text for prediction, the model doesn’t consider special text formatting and punctuation. For example, when you send the string “We had a great time! :)” to the model, the model returns a prediction for the string “We had a great time”.
- Batch predictions aren’t supported. When you send text in for a prediction, you make a single API call to the `/intent` endpoint or the `/sentiment` endpoint.
Improving a Model

Implementing a deep learning model is an iterative process. Continuing to refine your production model is part of the life cycle.

When you put a model into production, it’s a good idea to let users identify misclassified data as they send in data and get predictions. Creating a method to track misclassified data means that you can quickly get a model up and running. You can then continue to improve the model as you learn more about how it’s used and performs.

Implement a feedback loop in your apps with the feedback APIs. The high-level process includes these steps:

1. Build functionality in your app to identify image or text data that was misclassified.
2. Add the data, along with the correct label, to the dataset. Note that you add the feedback image or text to the dataset from which the model was created.
3. Train or retrain the dataset, and use the `withFeedback` training parameter.
4. Use the new or updated model after the feedback is incorporated.

Conclusion

Do you feel like a genius now? And just think, you didn’t need a degree.

Deep learning isn’t that difficult. It’s just a very iterative process. It’s all about making a really great plan, starting with a small use case, and then rolling it out and iterating until it’s almost perfect. Now, all that data you thought wasn’t useful, can be put to work in ways you never realized. You can go back to your colleagues and brag about how much more you know about deep learning and AI. Then, direct them to one of our modules so they can be geniuses too.

- Einstein Vision Module - sfdc.co/EinsteinVisionGSG
- Einstein Language Module - sfdc.co/EinsteinLanguageGSG
- Documentation or Einstein Platform Services Developer Guide - metamind.readme.io/
- Sign-up for a Free Trial - api.einstein.ai/signup
- Einstein Vision & Language Model Builder - sfdc.co/EinsteinVisionLanguageBuilder