Image Classification

Table of Contents

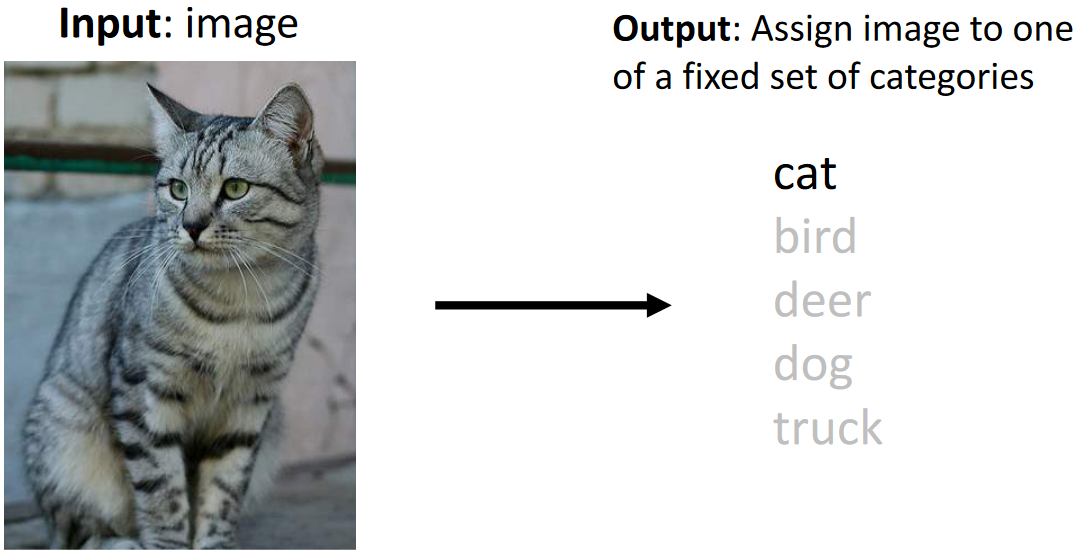
[Image Classifiers 4](#_Toc138881620)

[Nearest Neighbour Classifier 6](#_Toc138881621)

[Hyperparameters 9](#_Toc138881622)

[Data Splits 10](#_Toc138881623)

Once of the core computer vision tasks is **image classification**, where we take an input image and assign one of a fixed set of labels to the image.



The main problem with this task is the **semantic gap**. As humans, we can easily see that the picture is of a cat, but a computer does not process the image in the same manner that we do. The computer just sees a 2D matrix of numbers. This makes the task of identifying the cat difficult.

There are a variety of additional challenges associated with the task such as:

* **Viewpoint Variation** – It could perhaps be argued that a computer can memorize the values of the image, but this does not help. Taking the same picture again from a different angle will result in completely different numbers.
* **Interclass Variation** – Different cats can appear very different, resulting in a large number of completely different looking cats, which are all still cats nonetheless.
* **Fine-Grained Categories** – If we decide to classify different species of cats then the task becomes even more challenging, since the model must learn far more subtle features than just the general shape of a cat.



* **Background Clutter** – Background noise can make it far more difficult to identify the object.



* **Illumination Changes** – Variations in illuminations make it more difficult to identify that two images belong to the same class.
* **Deformation** – If the object is deformed, it can be difficult to identify the correct class.

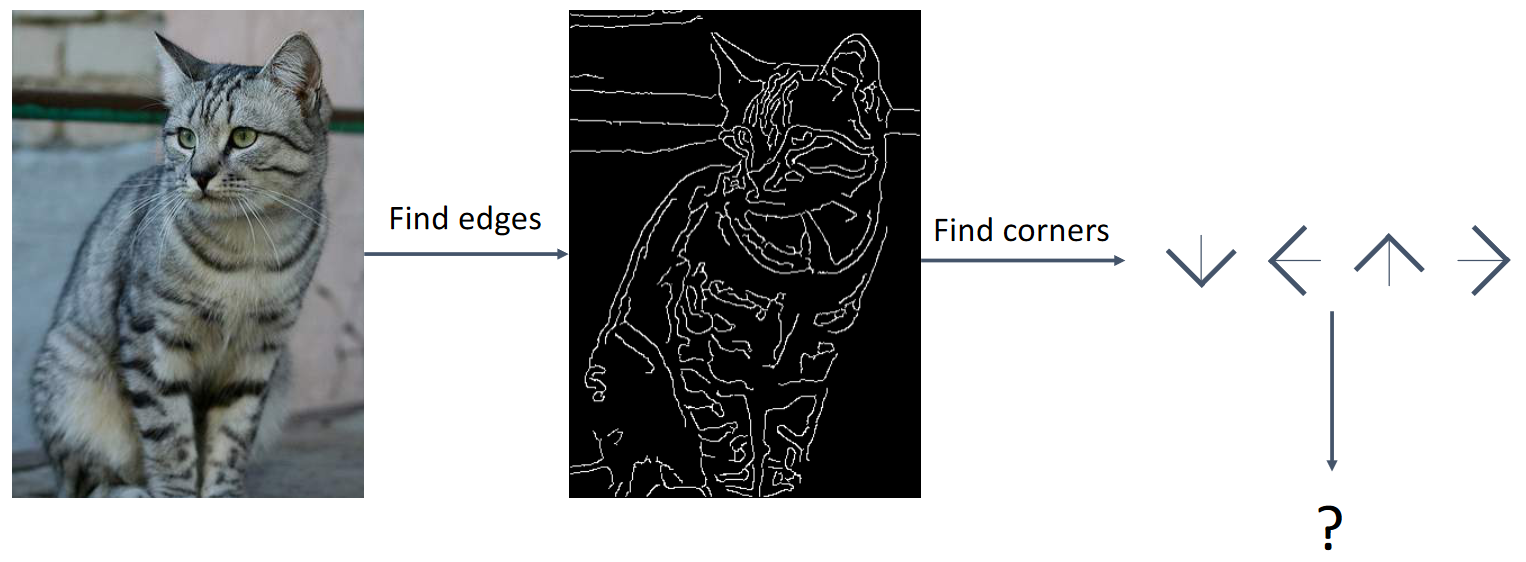


* **Occlusion** – If the object is partially covered (occluded), it can be difficult to identify the object.

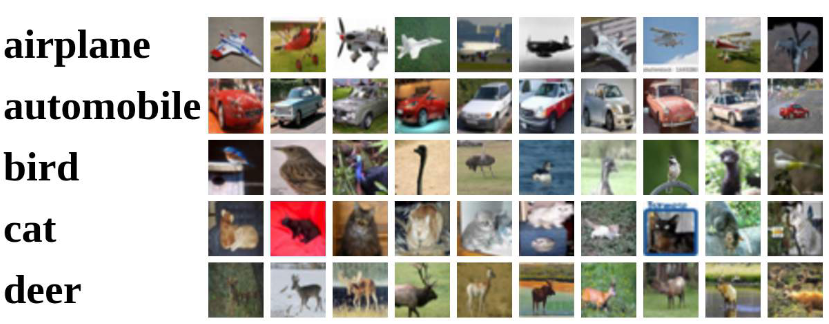
Despite these challenges, image classification is useful in a variety of tasks, such as medical imaging, galaxy classification or whale recognition. It is also used as a building block in other tasks like object detection, image captioning and even in games.

## Image Classifiers

An **image classifier** is an algorithm that takes an image as an input and returns a class label. Based on the challenges discussed above, it should be clear that there is no obvious way to hard code an algorithm to do this. One possibility, however, is to try detecting the edges and corners in the image. This is a method which has been traditionally used but is not very successful.



Modern machine learning approaches are all **data-driven**. We collect a dataset of images and their corresponding labels, train a classifier using machine learning and evaluate the classifier on new images.

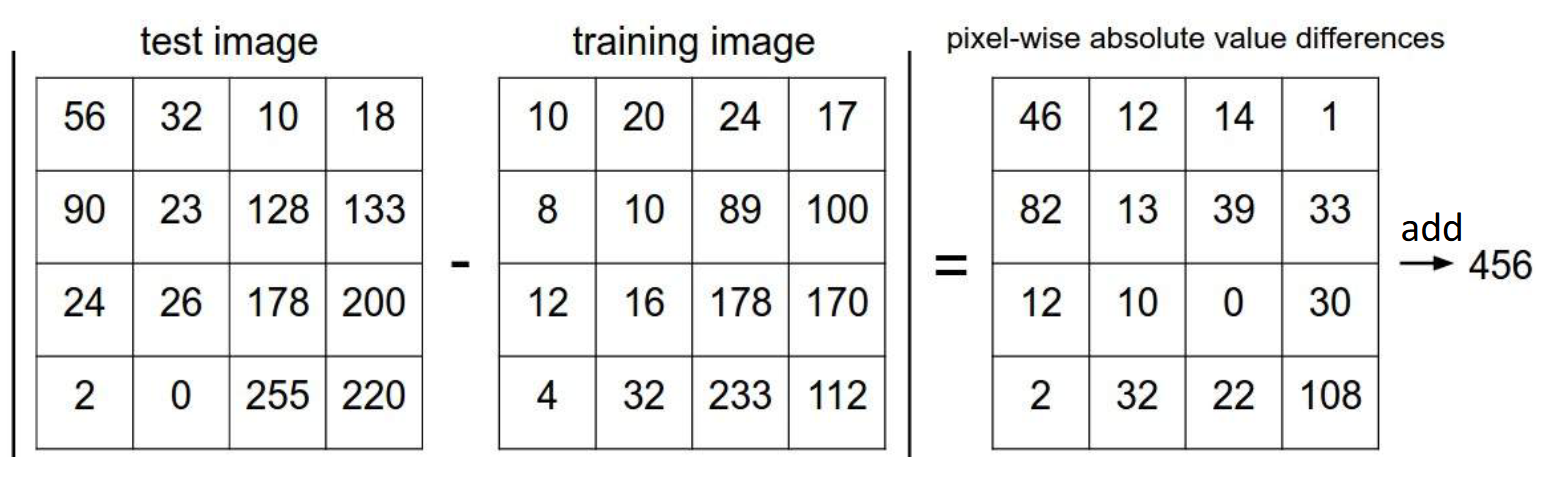


To this end, a variety of **image classification datasets** have been created that are used as benchmarks:

* **MNIST** – This is a relatively easy dataset of 50K training images and 10K test images. It consists of 28x28 grayscale images of handwritten digits from 0 to 9, meaning there are 10 classes. The results from these datasets are used as a starting point to evaluate new models. Their results are not representative of how the model will perform on more complex datasets.
* **CIFAR10** – This is a dataset of 32x32 RGB images of 10 classes. It has the same number of training and test images as MNIST. The additional challenge is that the images are slightly larger and are RGB instead of being grayscale.
* **CIFAR100** – This is the same as the CIFAR10 dataset except that there are 100 classes instead of 10. There are also 20 super-classes, each of which have 5 classes under them. Models can be evaluated on the 100 classes, which is a harder task, or the 20 super-classes, which is an easier task.
* **ImageNet** – This dataset consists of 1000 classes, 1.3 million training images, 50K validation images and 100K test images. Along with the normal evaluation metric of predicting the correct class, the dataset also allows for the ‘Top 5 Accuracy’, where the model is considered correct if one of its top 5 predictions for an image is correct. The images are of variable size but are usually resized to 256x256 during the training stage. There is also a variant of the dataset which has 22K categories.
* **MIT Places** – Unlike all the other datasets which concentrate on objects, this dataset has 365 classes of scenes. There are 8 million training images, 18.25K validation images and 328.5K test images. Like the ImageNet dataset, the images are of variable size and are often resized to 256x256 during training.
* **Omniglot** – This is an odd dataset with 1623 categories representing characters from 50 different languages. There are only 20 images per category. The limited number of images is used for a specific category of learning called few-shot learning, where the model is expected to learn patterns without using too much training data.

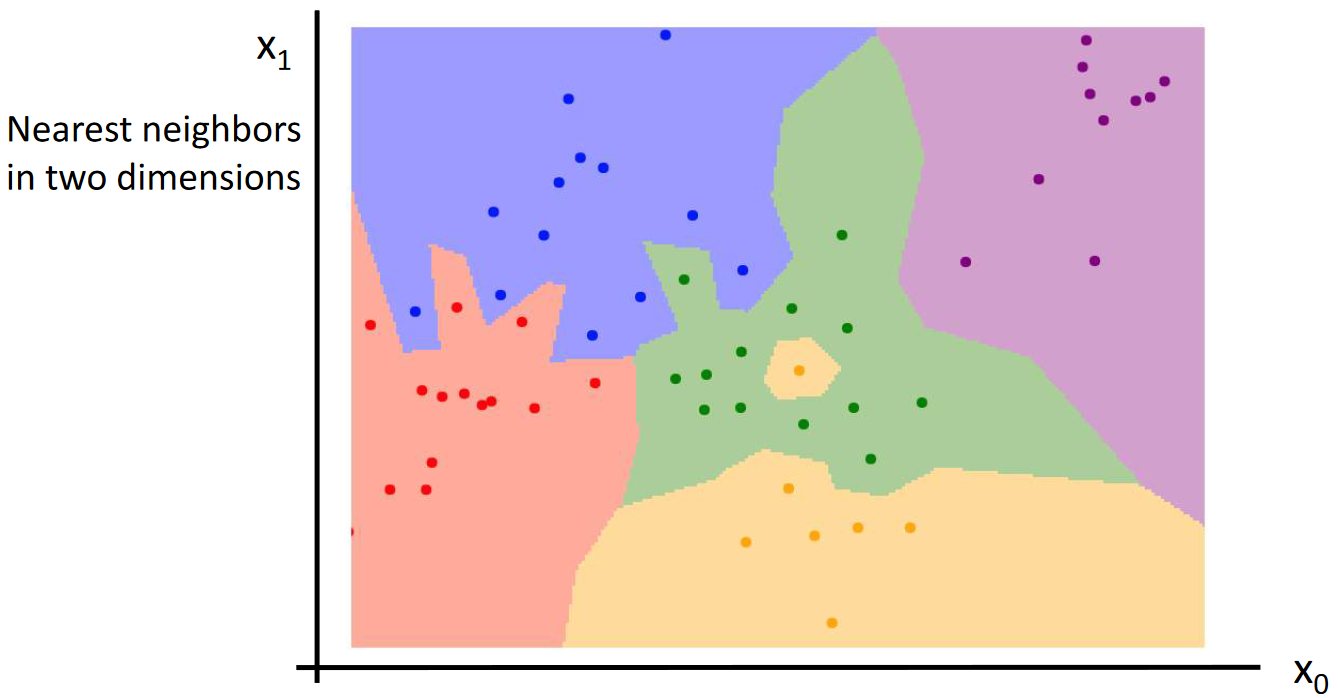
## Nearest Neighbour Classifier

The **Nearest Neighbour Classifier** is the simplest possible classifier. During the training stage, the classifier simply memorizes all the input data. For images, this is done by storing the values of the pixels in memory. In fact, this is not even ‘training’, since nothing is being learnt. During the test stage, the classifier finds the image that has the closest values (using the L1 or Manhattan distance metric). The corresponding label is considered to be the true label.

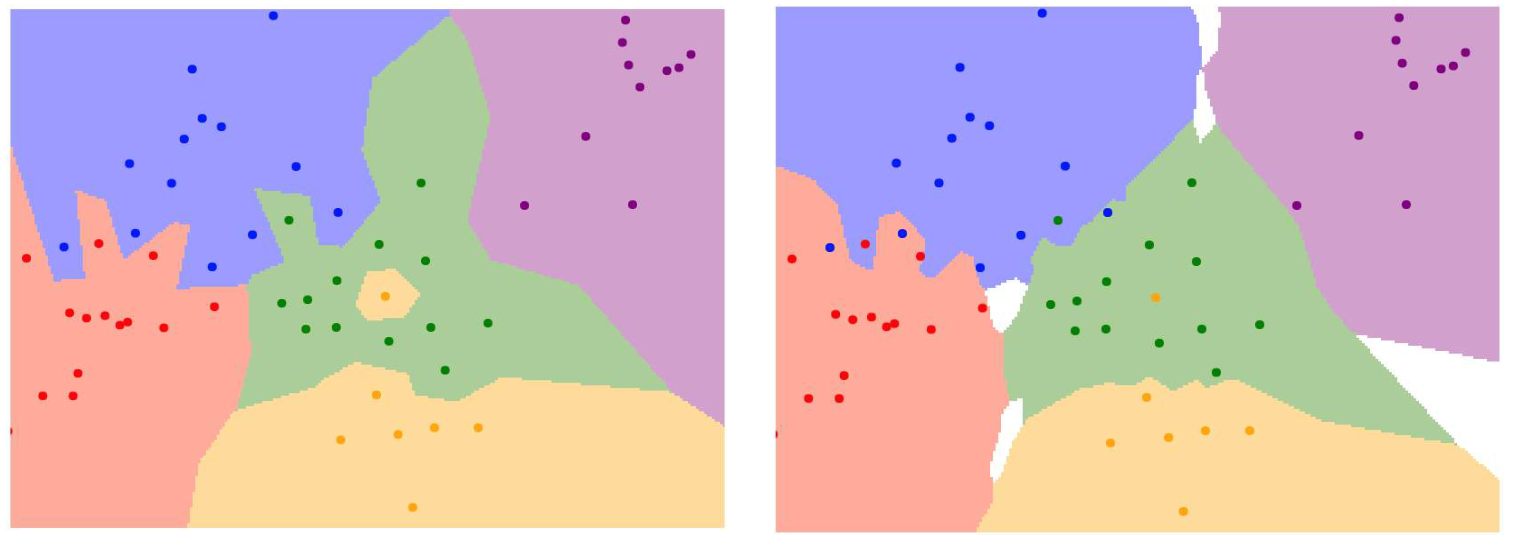


This classifier has a time complexity of O(1) during the training stage and O(N) during the test stage, N being the number of training images. This is the complete opposite of what we want from a classifier. It is essential that the testing stage take a very short amount of time for the classifier to be practical for use cases.

There are methods of approximating the nearest neighbour which decreases the amount of time required during the test stage, but there are other issues which make this a bad classifier as well. Since the classifier is not learning features but is just memorizing, it fails frequently due to any number of minor changes. Shifting the image, using an image that looks similar but is for a different class and adding noise to the image will all cause the classifier to fail. This happens because the **decision boundaries** the classifier draws are very tight, being affected by outliers.

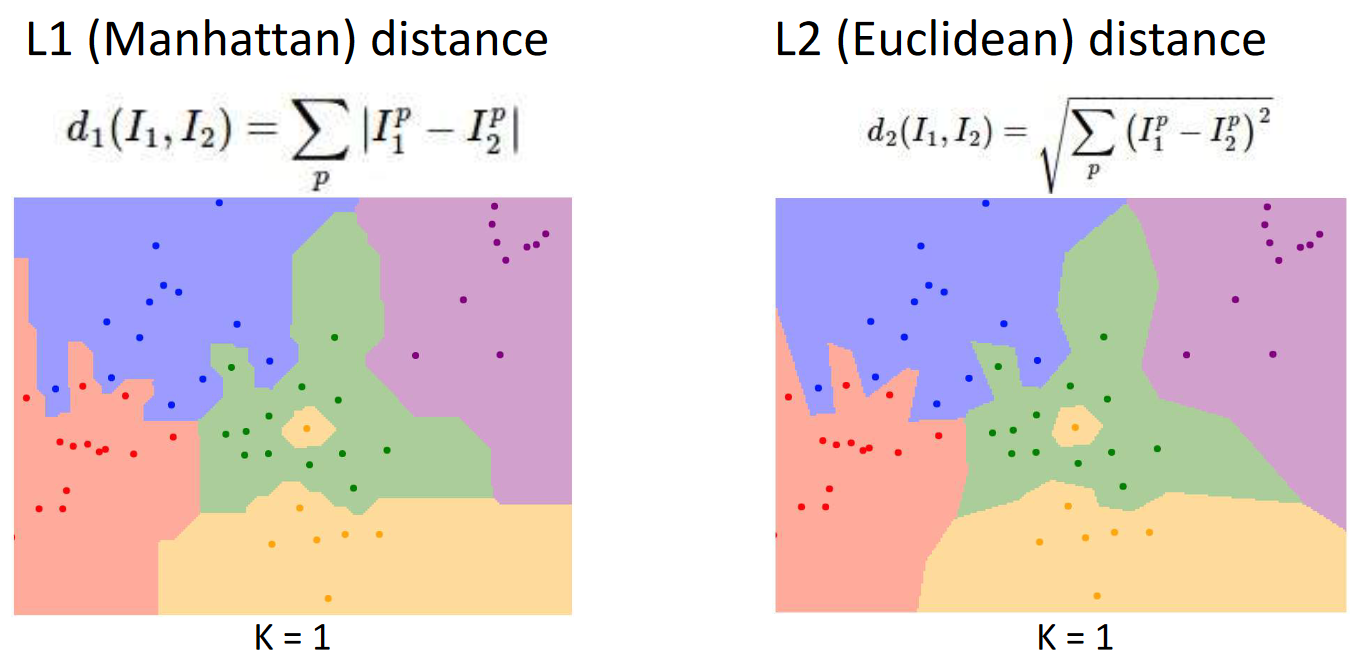


One way we can decrease this issue is to use multiple neighbours and take the majority vote. This mechanism is called the **K-Nearest Neighbour** (KNN) algorithm. For just three neighbours, the issue becomes much less severe.



The white regions in the graph on the right show areas where the number of votes for multiple classes were equal. We need to specify a mechanism for prioritizing classes in such cases.

Notice that despite using the KNN algorithm, we have a staircase effect near the boundaries. Not having smooth boundaries can also cause issues with the incorrect class being chosen. To smooth these out, we can use a different distance metric, perhaps the L2 (Euclidean) distance metric.



With the right choice of distance metric and value of , the KNN algorithm can actually be applied to any type of data. This, however, depends on the number of training samples the algorithm is given. As the number of training samples increases, the chances that a test sample will be very close to one of the training samples also increases, which in turn increases the accuracy of the algorithm. However, this comes at the cost of increased training time. Additionally, it is not as easy as it seems to have ‘enough’ training data. For just 32x32 black and white images the number of possible images is . This is more than number of particles in the visible universe, which basically means it is impossible to have enough training data to create a situation that we never see any new test data.

Because of such reasons, KNN is never really used directly. Instead, it works with other networks. For example, a different model like a ConvNet can be used to extract features from the raw pixels. Feeding these features to a KNN works really well, since new test images of the same categories will result in the ConvNet model given the exact same features to the KNN.

## Hyperparameters

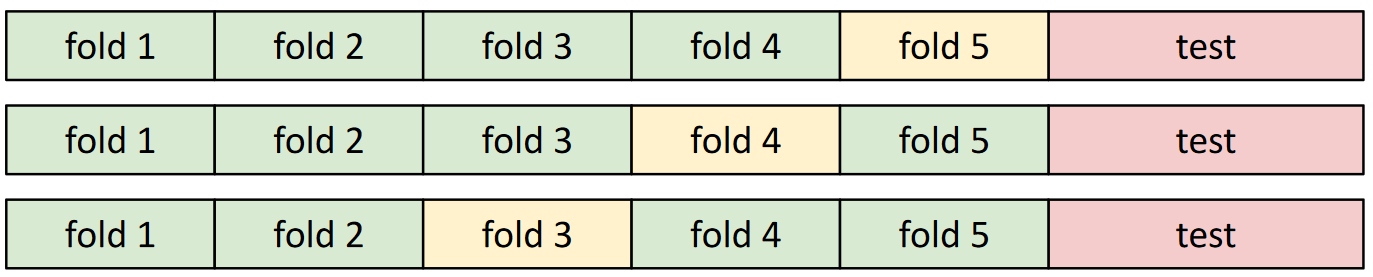
The value of and the distance metric are parameters that we choose rather than ones the model learns by itself. Such parameters are called **hyperparameters**. The values of different hyperparameters depend on the problem and cannot be suggested beforehand. The only way to find the correct ones is to use trial and error and perhaps intuition from previous experience.

## Data Splits

Simply training the model on the training data and giving the results as the final accuracy of the model is a poor way to judge the model because the model will always get 100% accuracy. To judge how well the model will do on new data, we need to have a separate **test set** which the model never sees. The performance of the model on that set is how the model will do in the real world.

However, even this is not enough. In reality, we will need to finetune a variety of hyperparameters to improve the performance of any model. We will change the hyperparameters based on how well the model is doing on the test set. By doing this, we are indirectly teaching the model about the features of the test set as well. This will again cause the model to not generalize to real life data. Instead, we should have yet another set of data called the **validation set**. We train the model and adjust hyperparameters based on how well it is doing on the validation set. Finally, once we are done training our model and are satisfied with it, we run it once on the validation set and report the results.

A variation of using a validation set is called **N-Fold Cross Validation**. The training and validation sets are rotated in groups called ‘folds’, as shown below. The average results over all combinations is considered to be the final validation set result.



This mechanism is useful when the amount of data we have is limited but is not usually used for deep-learning algorithms due to the increased computational cost of training the model multiple times.