

Lecture 07

Deep Learning 02

Roadmap to using DL for your image classification projects

2026-04-10

Sébastien Valade



This slide deck was originally created in **2021** (first course edition).

Some **Keras / TensorFlow API** details shown in the code examples may have changed since then.

1. Define a project
2. Label the data
3. Load & the data
4. Select the model
5. Train and predict

So far we've used datasets which were already structured for Tensor Flow

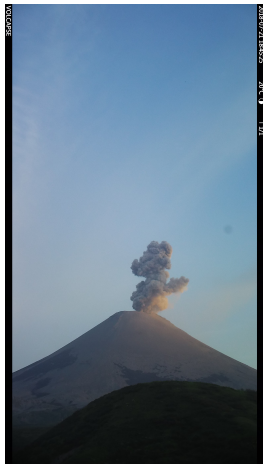
...

⇒ how do we handle our own dataset?

1. Define a project
2. Label the data
3. Load & the data
4. Select the model
5. Train and predict

1. Define a project

Project: *classify volcano web-camera images*



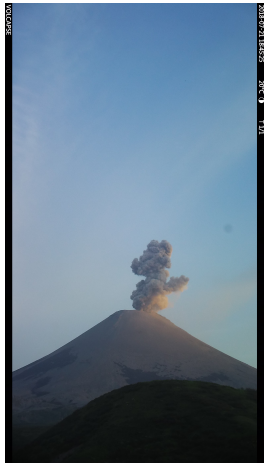
⇒ ash plume? gas plume? no visibility? night?

⇒ I have data and a problem to solve, now what?

1. label the data
2. load the data
3. select the model
4. train and evaluate!

1. Define a project

Project: *classify volcano web-camera images*



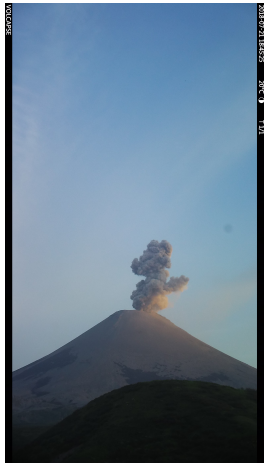
⇒ ash plume? gas plume? no visibility? night?

⇒ I have data and a problem to solve, now what?

1. label the data
2. load the data
3. select the model
4. train and evaluate!

1. Define a project

Project: *classify volcano web-camera images*



⇒ ash plume? gas plume? no visibility? night?

⇒ I have data and a problem to solve, now what?

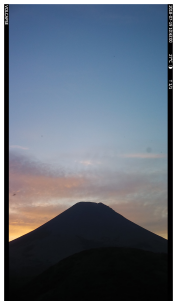
1. label the data
2. load the data
3. select the model
4. train and evaluate!

1. Define a project
2. Label the data
3. Load & the data
4. Select the model
5. Train and predict

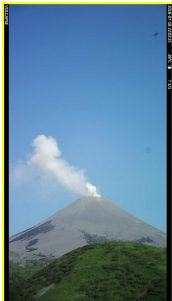
Label the data

⇒ go through your dataset, and **label** images from each class

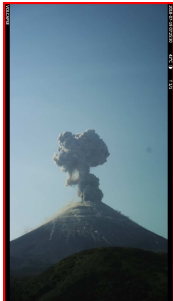
0



1



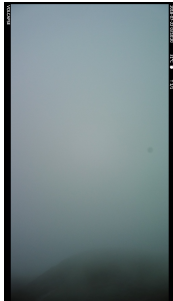
2



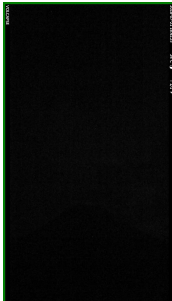
3



4



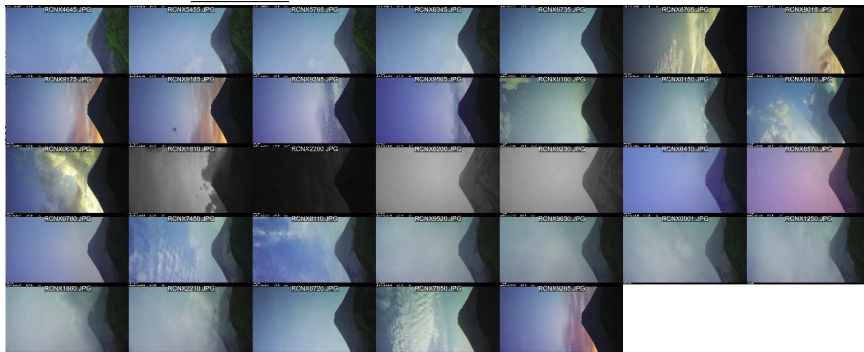
5



Label the data

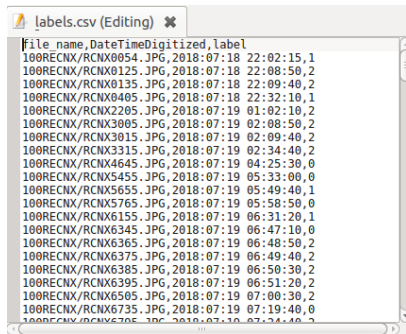
- ⇒ go through your dataset, and **label** images from each class
- ⇒ search for as much **variability** possible in each class

example: variability in *class "0" = no activity*



Label the data

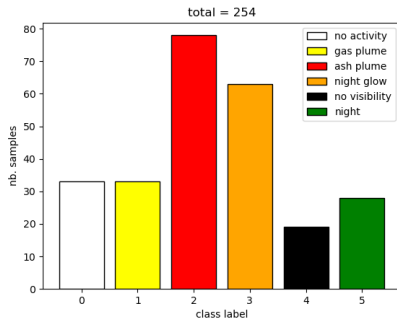
- ⇒ go through your dataset, and **label** images from each class
- ⇒ search for as much **variability** possible in each class
- ⇒ store the **file name**, **label**, and any additional information in a file (ex: .csv)



NB: depending on how your data structured, you can also choose to store images in distinct folders (one for each class)

Label the data

- ⇒ go through your dataset, and **label** images from each class
- ⇒ search for as much **variability** possible in each class
- ⇒ store the **file name**, **label**, and any additional information in a file (ex: .csv)
- ⇒ check the **distribution** of your samples for each class



NB: ideally should be equally distributed, but there are ways to overcome this (class weighting)

1. Define a project
2. Label the data
3. Load & the data
4. Select the model
5. Train and predict

Loading the data

- ⇒ use Tensor Flow's **Data API** to create and manipulate dataset object
- ⇒ shuffle your data, and split into train, validate, and test datasets
- ⇒ one way to do that:

```
# Load labels file
df = pd.read_csv("train.csv")

# Set indexes for train, test, validation datasets
n = len(df)
idx = np.arange(n) # Create array with n integers
np.random.seed(123) # Set seed to keep same randomization
np.random.shuffle(idx) # Modify sequence in-place by shuffling its contents

train_r, val_r, test_r = 0.8, 0.1, 0.1 # Set ratios for each dataset
idx_for_splitting = [int(n * train_r), int(n * (train_r+val_r))]
train_idx, val_idx, test_idx = np.split(idx, idx_for_splitting)

# Create tensor flow data set
file_names = df["file_name"].values
labels = df["label"].values

train_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[train_idx], labels[train_idx]))
val_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[val_idx], labels[val_idx]))
test_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[test_idx], labels[test_idx]))
```

Loading the data

⇒ use Tensor Flow's **Data API** to create and manipulate dataset object

⇒ **shuffle** your data, and **split** into train, validate, and test datasets

⇒ one way to do that:

```
# Load labels file
df = pd.read_csv("train.csv")

# Set indexes for train, test, validation datasets
n = len(df)
idx = np.arange(n) # Create array with n integers
np.random.seed(123) # Set seed to keep same randomization
np.random.shuffle(idx) # Modify sequence in-place by shuffling its contents

train_r, val_r, test_r = 0.8, 0.1, 0.1 # Set ratios for each dataset
idx_for_splitting = [int(n * train_r), int(n * (train_r+val_r))]
train_idx, val_idx, test_idx = np.split(idx, idx_for_splitting)

# Create tensor flow data set
file_names = df["file_name"].values
labels = df["label"].values

train_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[train_idx], labels[train_idx]))
val_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[val_idx], labels[val_idx]))
test_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[test_idx], labels[test_idx]))
```


Loading the data

- ⇒ use Tensor Flow's **Data API** to create and manipulate dataset object
- ⇒ **shuffle** your data, and **split** into train, validate, and test datasets
- ⇒ one way to do that:

```
# Load labels file
df = pd.read_csv("train.csv")

# Set indexes for train, test, validation datasets
n = len(df)
idx = np.arange(n) # Create array with n integers
np.random.seed(123) # Set seed to keep same randomization
np.random.shuffle(idx) # Modify sequence in-place by shuffling its contents

train_r, val_r, test_r = 0.8, 0.1, 0.1 # Set ratios for each dataset
idx_for_splitting = [int(n * train_r), int(n * (train_r+val_r))]
train_idx, val_idx, test_idx = np.split(idx, idx_for_splitting)

# Create tensor flow data set
file_names = df["file_name"].values
labels = df["label"].values

train_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[train_idx], labels[train_idx]))
val_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[val_idx], labels[val_idx]))
test_ds_raw = tf.data.Dataset.from_tensor_slices((file_names[test_idx], labels[test_idx]))
```

Prepare the data

At this stage, the datasets generated are *TensorSliceDataset* objects, storing *filename* and *label*:

```
for file_name, label in iter(train_ds_raw):
    print('---')
    print(file_name)
    print(label)

# Returns:
# ---
# tf.Tensor(b'104REC NX/RCNX3406.JPG', shape=(), dtype=string)
# tf.Tensor(3.0, shape=(), dtype=float64)
# ---
# tf.Tensor(b'100REC NX/RCNX6795.JPG', shape=(), dtype=string)
# tf.Tensor(2.0, shape=(), dtype=float64)
# ...
```

⇒ we now need to “instruct” which operations these datasets should undergo during training

⇒ dataset objects allow to chain transformations easily: *map functions, define batch, etc.*

Prepare the data

At this stage, the datasets generated are *TensorSliceDataset* objects, storing *filename* and *label*:

```
for file_name, label in iter(train_ds_raw):
    print('---')
    print(file_name)
    print(label)

# Returns:
# ---
# tf.Tensor(b'104RECNX/RCNX3406.JPG', shape=(), dtype=string)
# tf.Tensor(3.0, shape=(), dtype=float64)
# ---
# tf.Tensor(b'100RECNX/RCNX6795.JPG', shape=(), dtype=string)
# tf.Tensor(2.0, shape=(), dtype=float64)
# ...
```

⇒ we now need to “instruct” which operations these datasets should undergo during training

⇒ dataset objects allow to chain transformations easily: *map functions, define batch, etc.*

Prepare the data

At this stage, the datasets generated are *TensorSliceDataset* objects, storing *filename* and *label*:

```
for file_name, label in iter(train_ds_raw):
    print('---')
    print(file_name)
    print(label)

# Returns:
# ---
# tf.Tensor(b'104RECNX/RCNX3406.JPG', shape=(), dtype=string)
# tf.Tensor(3.0, shape=(), dtype=float64)
# ---
# tf.Tensor(b'100RECNX/RCNX6795.JPG', shape=(), dtype=string)
# tf.Tensor(2.0, shape=(), dtype=float64)
# ...
```

⇒ we now need to “instruct” which operations these datasets should undergo during training

⇒ dataset objects allow to **chain transformations** easily: *map functions, define batch, etc.*

Prepare the data

⇒ chain transformations:

```
resize_h, resize_w = 130, 230

def preprocess(image_file, label):

    # Read image
    image = tf.io.read_file(path_root + image_file)
    image = tf.image.decode_jpeg(image, channels=3) # returns uint8 tensor

    # Convert to float to prepare for resize
    image = tf.image.convert_image_dtype(image, tf.float32)

    # Resize image (original size/10)
    # => returns float [0-1]
    resized_image = tf.image.resize(image,
                                    size=(resize_h, resize_w), # (new_height, new_width)
                                    preserve_aspect_ratio=True)

    # Xception preprocess_input:
    # => input: floating point with values in range [0, 255] (doc)
    # => returns scaled input pixels between -1 and 1 (https://keras.io/api/applications/xception/)
    resized_image = tf.multiply(resized_image, 255) # switch to range 0-255
    final_image = tf.keras.applications.xception.preprocess_input(resized_image) # not clear how to give inputs (dtype/range)

    return final_image, label

batch_size = 32
train_ds = train_ds_raw.shuffle(buffer_size=1000, seed=None) # => at each epoch training will see samples in different order
train_ds = train_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
val_ds = val_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
test_ds = test_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
```

Prepare the data

⇒ chain transformations:

```
resize_h, resize_w = 130, 230

def preprocess(image_file, label):

    # Read image
    image = tf.io.read_file(path_root + image_file)
    image = tf.image.decode_jpeg(image, channels=3) # returns uint8 tensor

    # Convert to float to prepare for resize
    image = tf.image.convert_image_dtype(image, tf.float32)

    # Resize image (original size/10)
    # => returns float [0-1]
    resized_image = tf.image.resize(image,
                                    size=(resize_h, resize_w), # (new_height, new_width)
                                    preserve_aspect_ratio=True)

    # Xception preprocess_input:
    # => input: floating point with values in range [0, 255] (doc)
    # => returns scaled input pixels between -1 and 1 (https://keras.io/api/applications/xception/)
    resized_image = tf.multiply(resized_image, 255) # switch to range 0-255
    final_image = tf.keras.applications.xception.preprocess_input(resized_image) # not clear how to give inputs (dtype/range)

    return final_image, label

batch_size = 32
train_ds = train_ds_raw.shuffle(buffer_size=1000, seed=None) # => at each epoch training will see samples in different order
train_ds = train_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
val_ds = val_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
test_ds = test_ds_raw.map(preprocess).batch(batch_size).prefetch(1)
```

1. Define a project
2. Label the data
3. Load & the data
- 4. Select the model**
5. Train and predict

Select the model

⇒ when you have a small amount of labeled data, choose a transfer learning solution

Code from last week exercise:

```
# Load Xception model and define as base model
base_model = tf.keras.applications.xception.Xception(weights="imagenet",
                                                    include_top=False,
                                                    input_shape=(resize_h, resize_w, 3))

# Freeze base layers
for layer in base_model.layers:
    layer.trainable = False

# Add layers to train classifier
avg = tf.keras.layers.GlobalAveragePooling2D()(base_model.output) # takes base_model outputs as input
output = tf.keras.layers.Dense(n_classes, activation="softmax")(avg) # takes GlobalAveragePooling2D layer as input

# Create final model
model = tf.keras.Model(inputs=base_model.input, outputs=output)

# Compile
optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```


1. Define a project
2. Label the data
3. Load & the data
4. Select the model
5. Train and predict

Train

Recall our dataset does not have a *uniform class distribution*

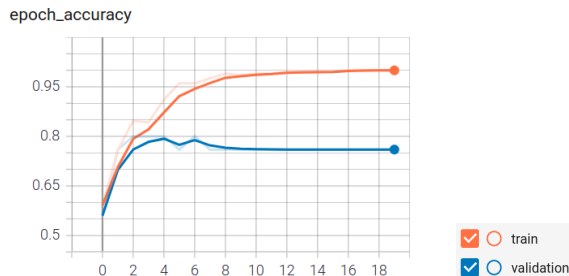
⇒ we can compensate for that using the `class_weight` option:

```
# Calculate class weights
labels_int = labels.astype('int64')
class0_nb, class1_nb, class2_nb, class3_nb, class4_nb, class5_nb = np.bincount(labels_int)
scaling_factor = n_samples / n_classes
class_weight = {0: (1 / class0_nb) * scaling_factor,
                1: (1 / class1_nb) * scaling_factor,
                2: (1 / class2_nb) * scaling_factor,
                3: (1 / class3_nb) * scaling_factor,
                4: (1 / class4_nb) * scaling_factor,
                5: (1 / class5_nb) * scaling_factor
               }

# Train
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, update_freq='batch')
history = model.fit(train_ds,
                    epochs=epochs,
                    validation_data=val_ds,
                    callbacks=[tensorboard_callback],
                    class_weight=class_weight)
```

Train

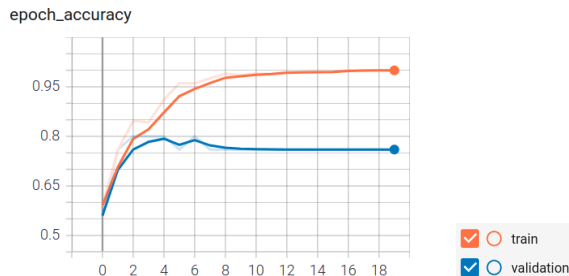
⇒ track accuracy of *training* & *validation* datasets with TensorBoard:



NB: there's room for improvement! e.g., more training data, data augmentation, regularization, fine-tuning, etc.

Train

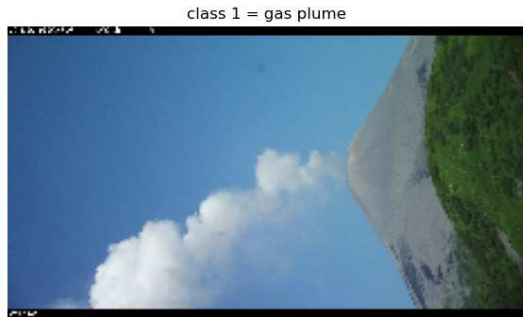
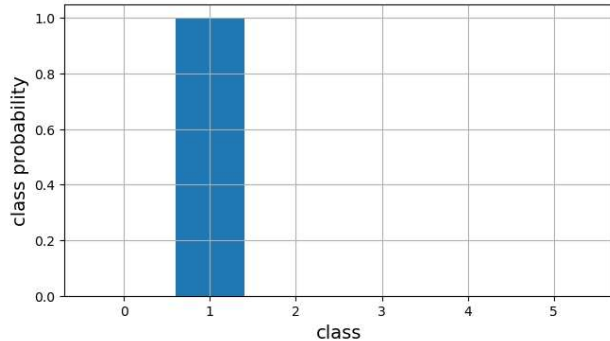
⇒ track accuracy of *training* & *validation* datasets with TensorBoard:



NB: there's room for improvement! e.g., more training data, data augmentation, regularization, fine-tuning, etc.

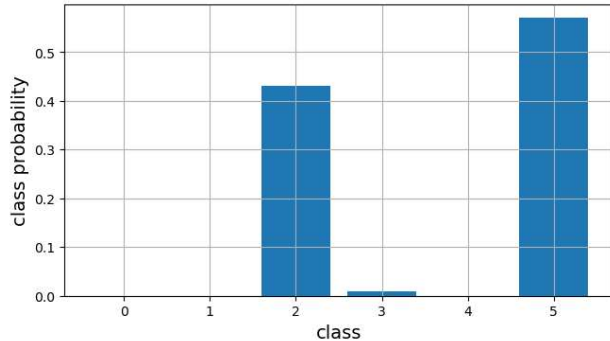
Predict

⇒ how well is our model predicting the test dataset?



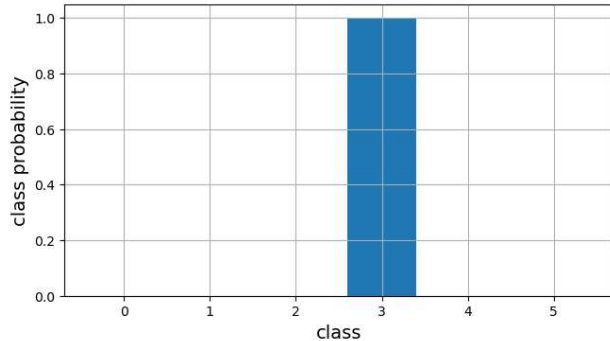
Predict

⇒ how well is our model predicting the test dataset?



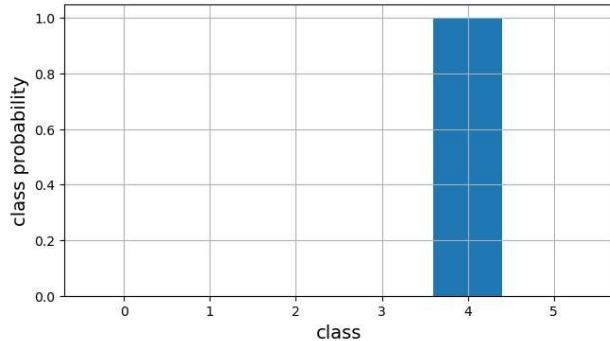
Predict

⇒ how well is our model predicting the test dataset?



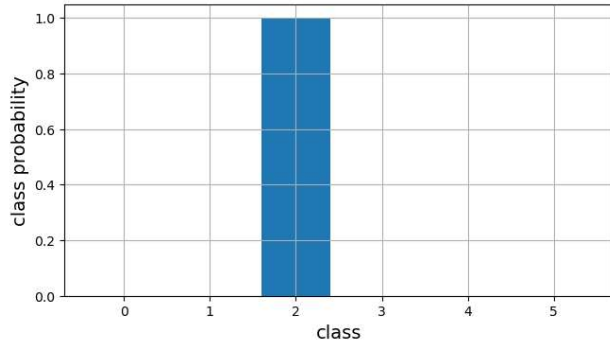
Predict

⇒ how well is our model predicting the test dataset?



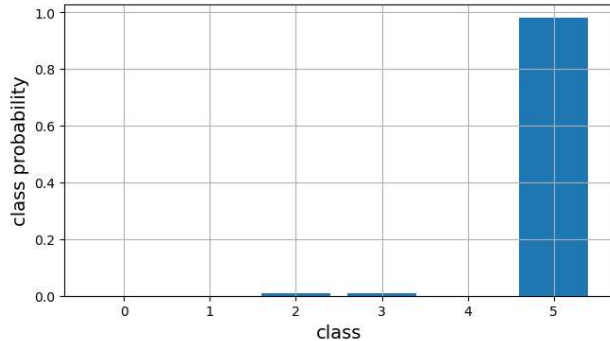
Predict

⇒ how well is our model predicting the test dataset?



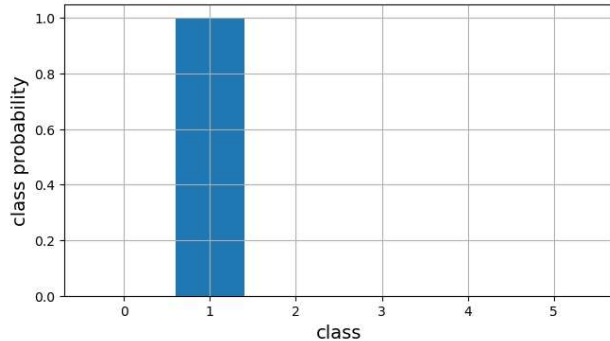
Predict

⇒ how well is our model predicting the test dataset?



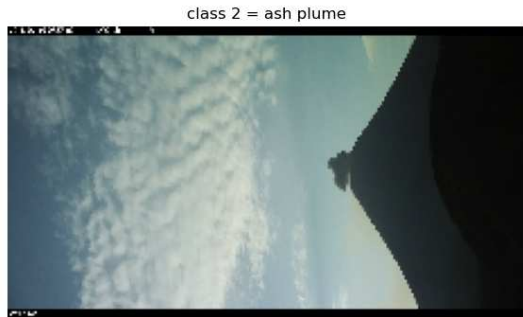
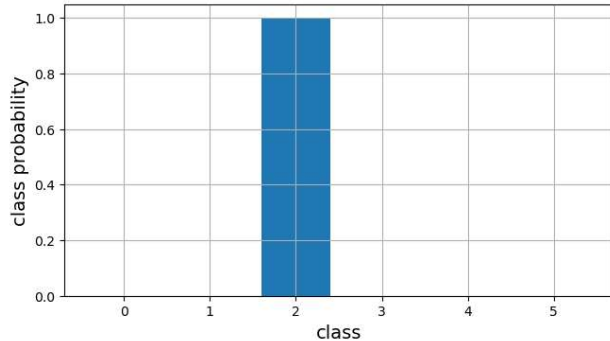
Predict

⇒ how well is our model predicting the test dataset?



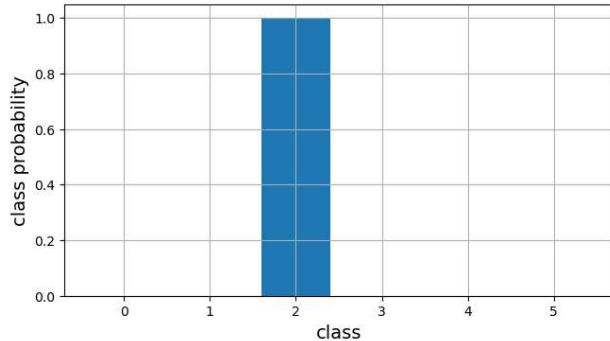
Predict

⇒ how well is our model predicting the test dataset?



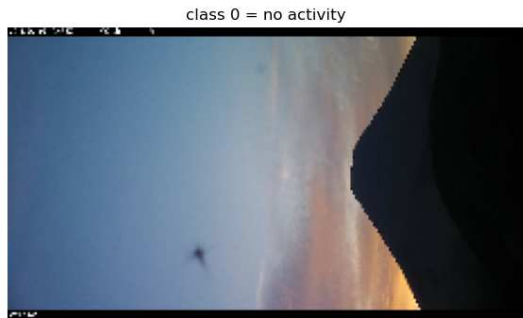
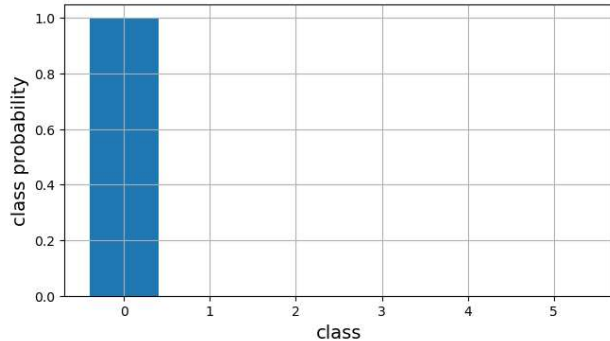
Predict

⇒ how well is our model predicting the test dataset?



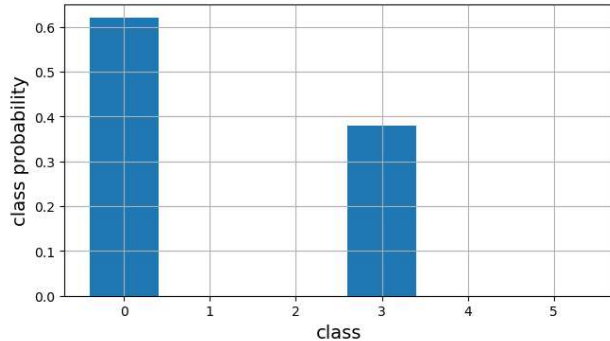
Predict

⇒ how well is our model predicting the test dataset?



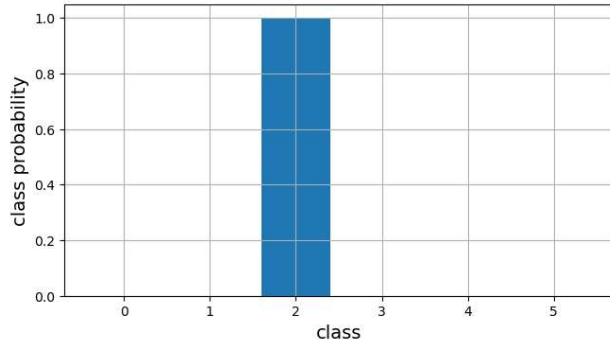
Predict

⇒ how well is our model predicting the test dataset?



Predict

⇒ how well is our model predicting the test dataset?



⇒ not bad for such little training dataset and time to train the model

⇒ but need to increase the number & diversity of the training images to avoid overfitting!

⇒ not bad for such little training dataset and time to train the model

⇒ but need to increase the number & diversity of the training images to avoid overfitting!