Code Error Detection

Shvetank Prakash
SP3816
Undergraduate in Computer Engineering
MECE 3998
Summer 2021 A

Aim & Project Idea

“The industry average is between 15 and 50 bugs per 1,000 lines of code…”

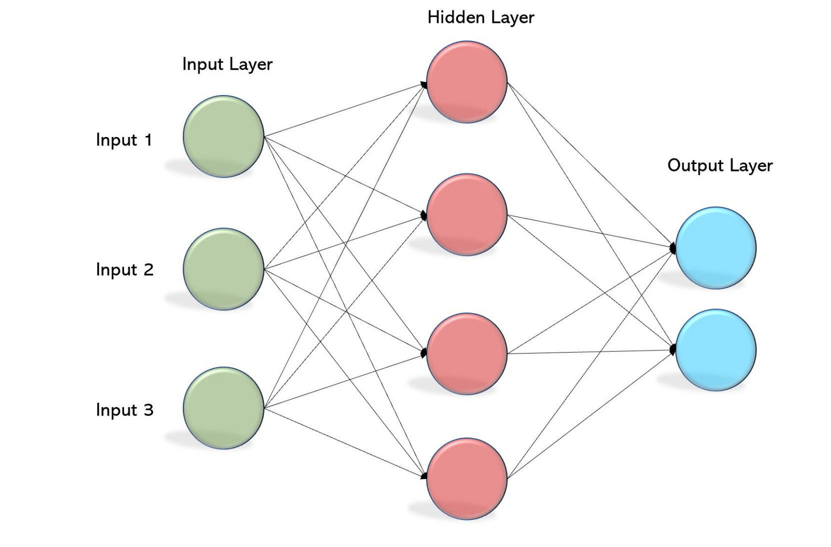
We want to see if we can train a DL system to spot bugs in code:

1. Download a ton of Python or C++ code from GitHub (choose one language).
2. Automatically generate a few million samples by:
   1. Taking screenshot of some random code page.
   2. Make one character change.
   3. Take a second screenshot.
3. Use the sample of good and bad code, see if we can train a system to discriminate with above 50% accuracy. Anything better than 50% is interesting.

Steps Taken & Ideas Investigated

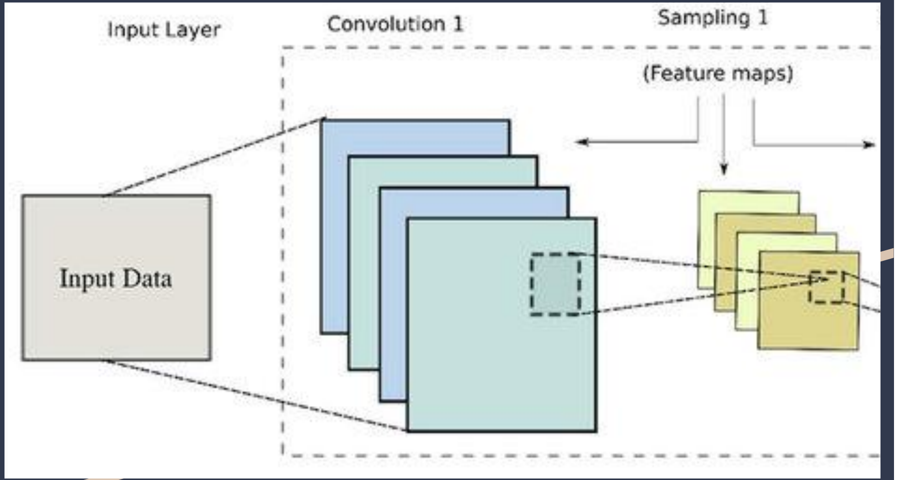
1. Create Dataset
2. MLP
3. CNN
4. Transfer Learning and Fine Tuning (ResNet)
5. Anomaly Detection (Autoencoder)
6. Dataset Revision
7. Summary of Findings

Multi-Layer Perceptron

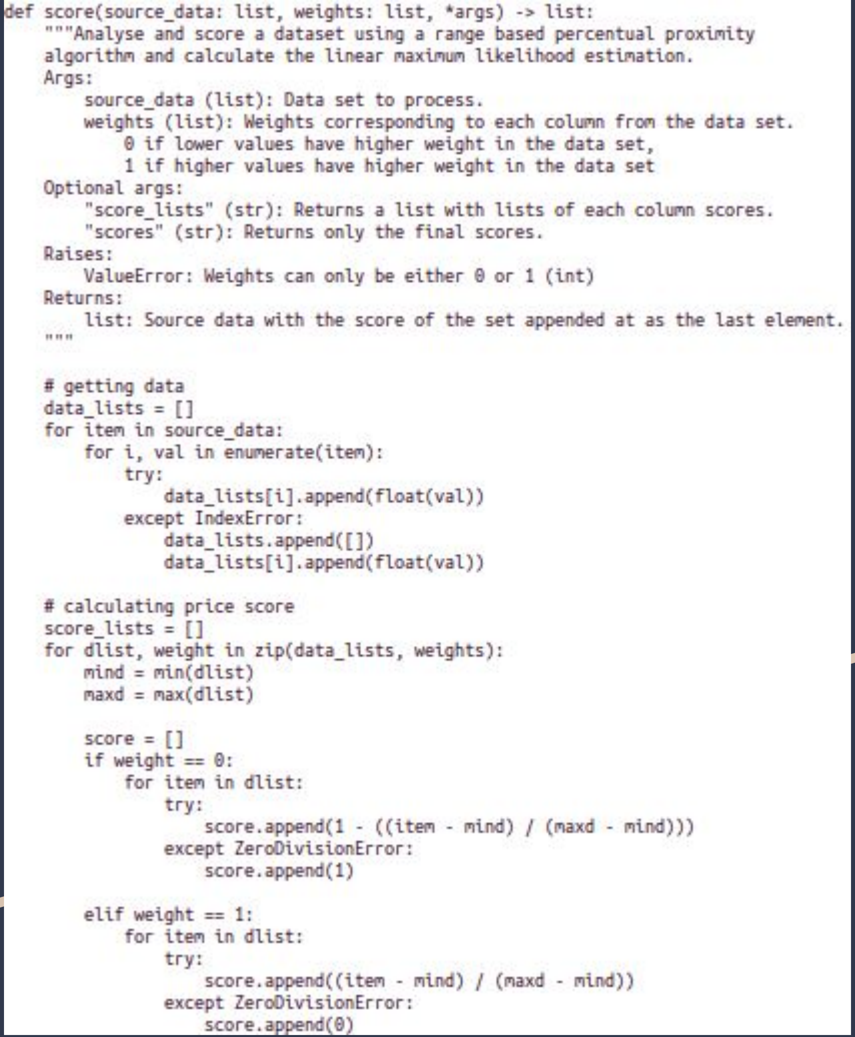


- Initial attempt to get some preliminary results and baseline for comparison
- Obtained 55.94% accuracy
- Solid start (unexpected)
  - Somewhat “deceived”

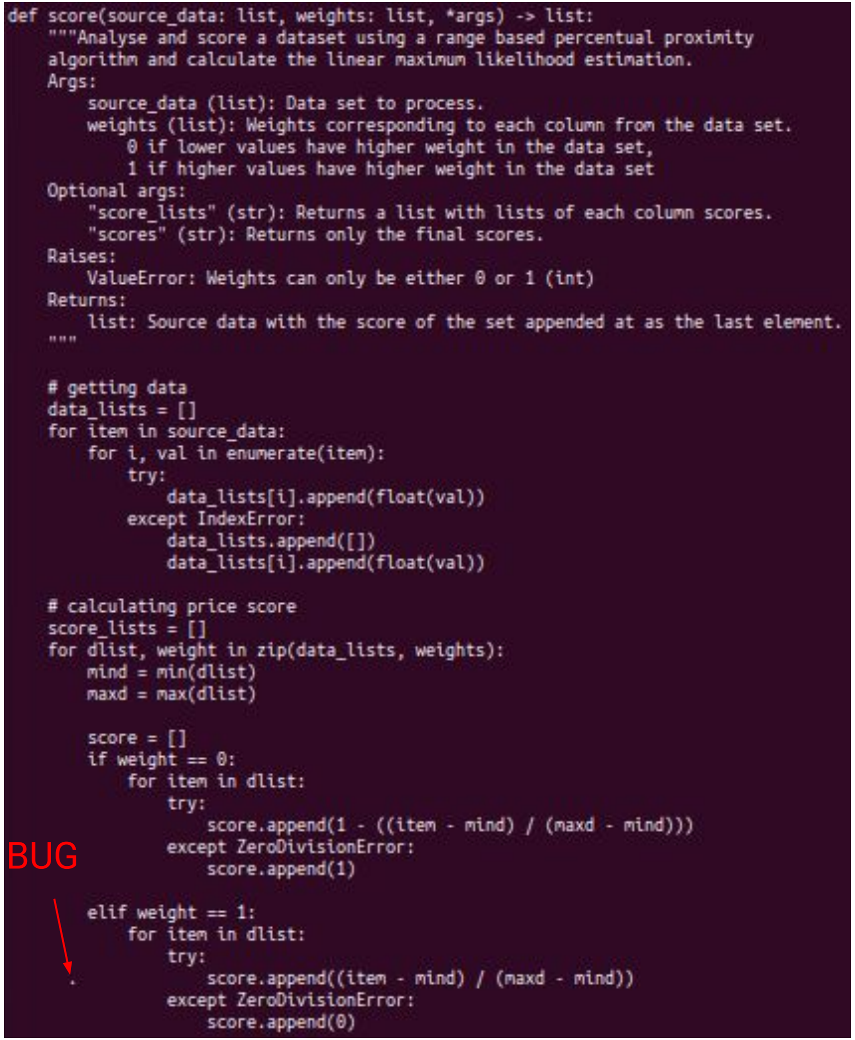
Convolutional Neural Network



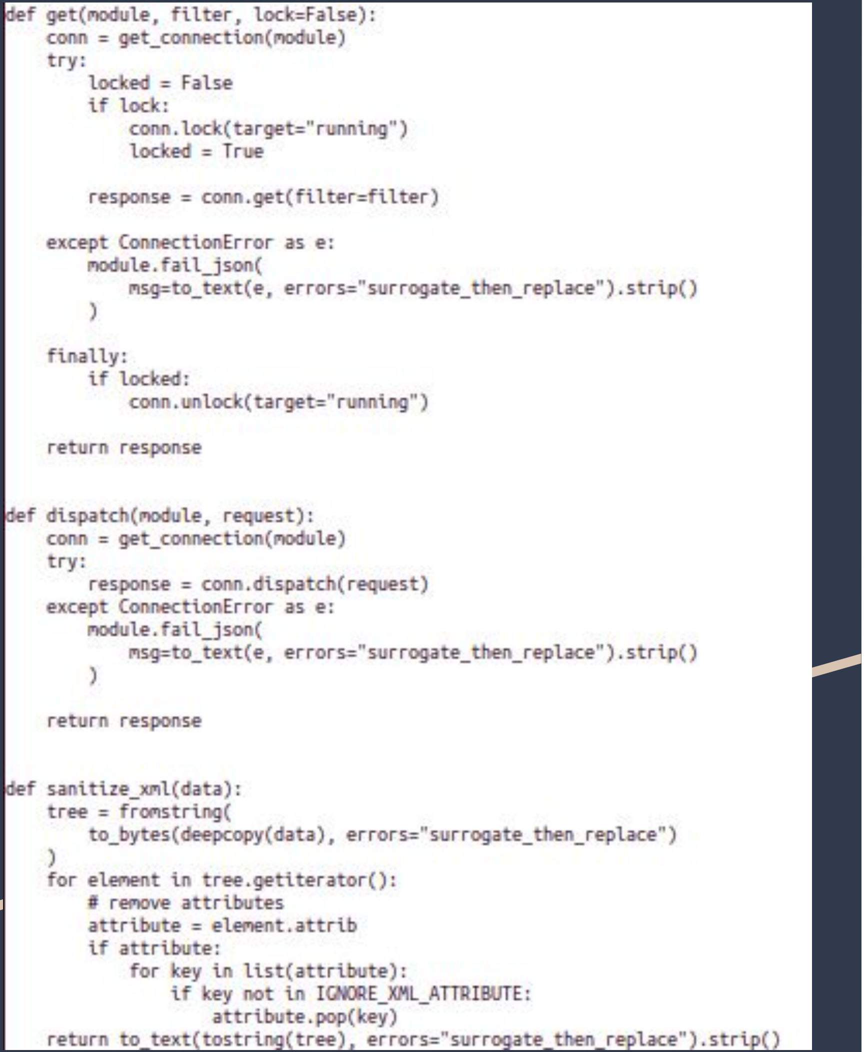
- After MLP results, tested with more complex model: CNN
- Obtained 91.86% accuracy
- Such great accuracy made me question results
    - Investigated types of errors in code (see next slide)
- Limitation of bug type not ideal → refined dataset
- Accuracy dropped and could not get > 50% with CNN



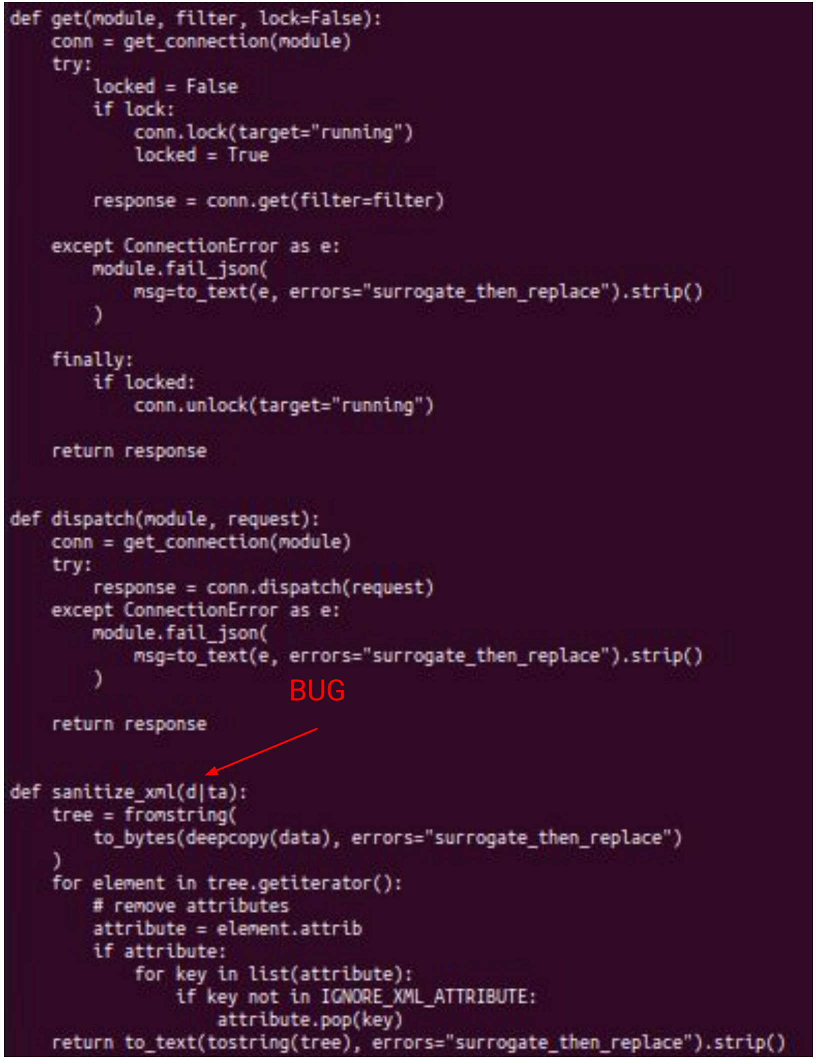
```
def score(source_data: list, weights: list, *args) -> list:
    """Analyse and score a dataset using a range based percentual proximity
    algorithm and calculate the linear maximum likelihood estimation.
    Args:
        source_data (list): Data set to process.
        weights (list): Weights corresponding to each column from the data set.
            0 if lower values have higher weight in the data set,
            1 if higher values have higher weight in the data set
    Optional args:
        "score_lists" (str): Returns a list with lists of each column scores.
        "scores" (str): Returns only the final scores.
    Raises:
        ValueError: Weights can only be either 0 or 1 (int)
    Returns:
        list: Source data with the score of the set appended at as the last element.
    """

    # getting data
    data_lists = []
    for item in source_data:
        for i, val in enumerate(item):
            try:
                data_lists[i].append(float(val))
            except IndexError:
                data_lists.append([])
                data_lists[i].append(float(val))

    # calculating price score
    score_lists = []
    for dlist, weight in zip(data_lists, weights):
        mind = min(dlist)
        maxd = max(dlist)

        score = []
        if weight == 0:
            for item in dlist:
                try:
                    score.append(1 - ((item - mind) / (maxd - mind)))
                except ZeroDivisionError:
                    score.append(1)

        elif weight == 1:
            for item in dlist:
                try:
                    score.append((item - mind) / (maxd - mind))
                except ZeroDivisionError:
                    score.append(0)
```



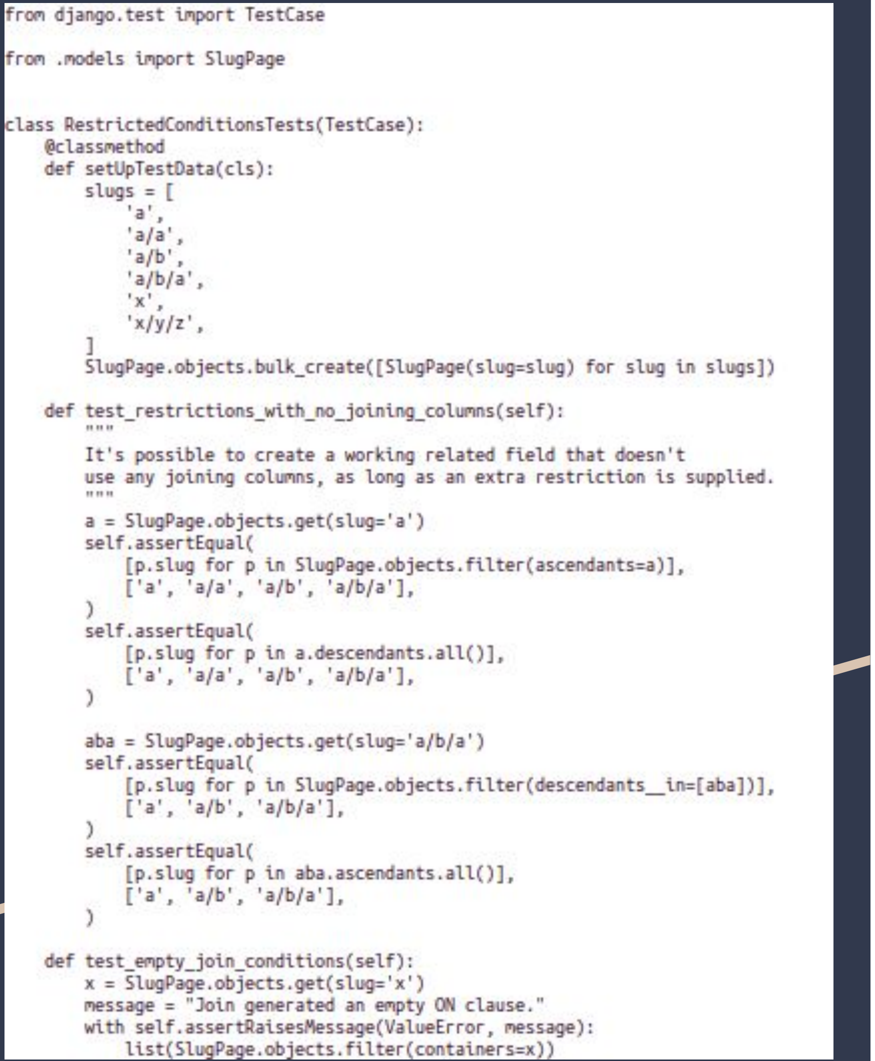
```
def score(source_data: list, weights: list, *args) -> list:
    """Analyse and score a dataset using a range based percentual proximity
    algorithm and calculate the linear maximum likelihood estimation.
    Args:
        source_data (list): Data set to process.
        weights (list): Weights corresponding to each column from the data set.
            0 if lower values have higher weight in the data set,
            1 if higher values have higher weight in the data set
    Optional args:
        "score_lists" (str): Returns a list with lists of each column scores.
        "scores" (str): Returns only the final scores.
    Raises:
        ValueError: Weights can only be either 0 or 1 (int)
    Returns:
        list: Source data with the score of the set appended at as the last element.
    """

    # getting data
    data_lists = []
    for item in source_data:
        for i, val in enumerate(item):
            try:
                data_lists[i].append(float(val))
            except IndexError:
                data_lists.append([])
                data_lists[i].append(float(val))

    # calculating price score
    score_lists = []
    for dlist, weight in zip(data_lists, weights):
        mind = min(dlist)
        maxd = max(dlist)

        score = []
        if weight == 0:
            for item in dlist:
                try:
                    score.append(1 - ((item - mind) / (maxd - mind)))
                except ZeroDivisionError:
                    score.append(1)

        elif weight == 1:
            for item in dlist:
                try:
                    score.append((item - mind) / (maxd - mind))
                except ZeroDivisionError:
                    score.append(0)
```



```
def get(module, filter, lock=False):
    conn = get_connection(module)
    try:
        locked = False
        if lock:
            conn.lock(target="running")
            locked = True

        response = conn.get(filter=filter)

    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    finally:
        if locked:
            conn.unlock(target="running")

    return response


def dispatch(module, request):
    conn = get_connection(module)
    try:
        response = conn.dispatch(request)
    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    return response


def sanitize_xml(data):
    tree = fromstring(
        to_bytes(deepcopy(data), errors="surrogate_then_replace")
    )
    for element in tree.getiterator():
        # remove attributes
        attribute = element.attrib
        if attribute:
            for key in list(attribute):
                if key not in IGNORE_XML_ATTRIBUTE:
                    attribute.pop(key)
    return to_text(tostring(tree), errors="surrogate_then_replace").strip()
```

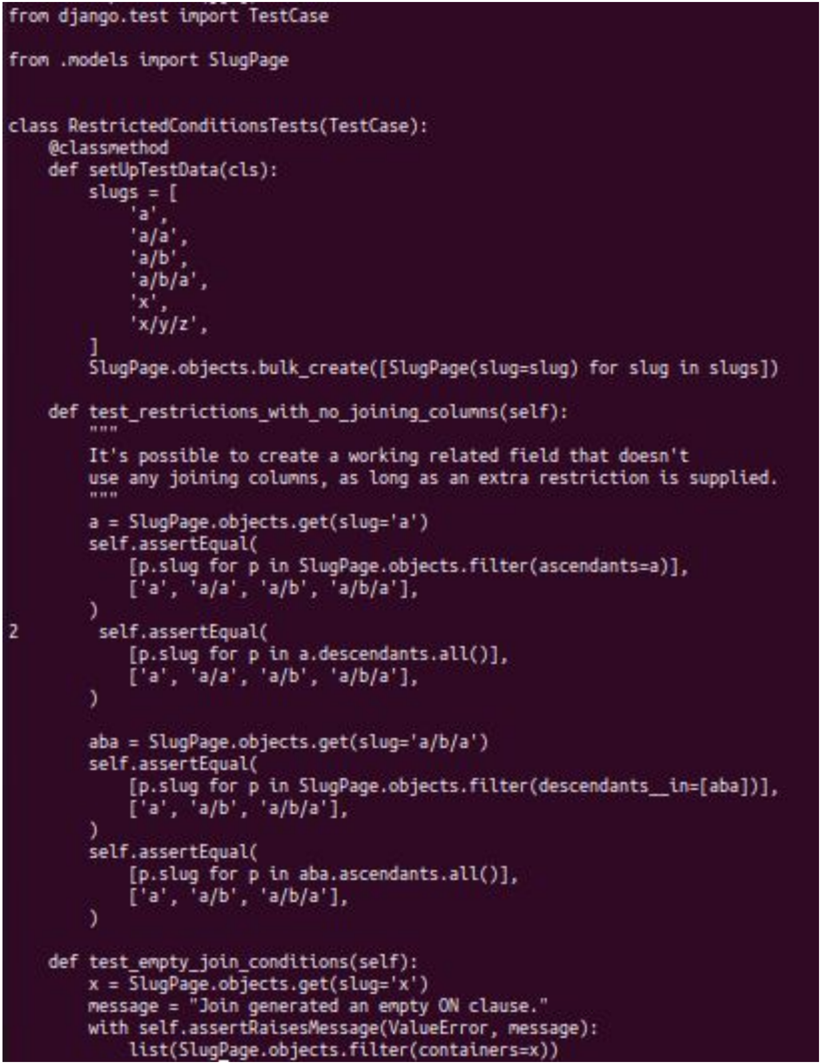


```
def get(module, filter, lock=False):
    conn = get_connection(module)
    try:
        locked = False
        if lock:
            conn.lock(target="running")
            locked = True

        response = conn.get(filter=filter)

    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    finally:
        if locked:
            conn.unlock(target="running")

    return response


def dispatch(module, request):
    conn = get_connection(module)
    try:
        response = conn.dispatch(request)
    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    return response


def sanitize_xml(d|ta):
    tree = fromstring(
        to_bytes(deepcopy(data), errors="surrogate_then_replace")
    )
    for element in tree.getiterator():
        # remove attributes
        attribute = element.attrib
        if attribute:
            for key in list(attribute):
                if key not in IGNORE_XML_ATTRIBUTE:
                    attribute.pop(key)
    return to_text(tostring(tree), errors="surrogate_then_replace").strip()
```

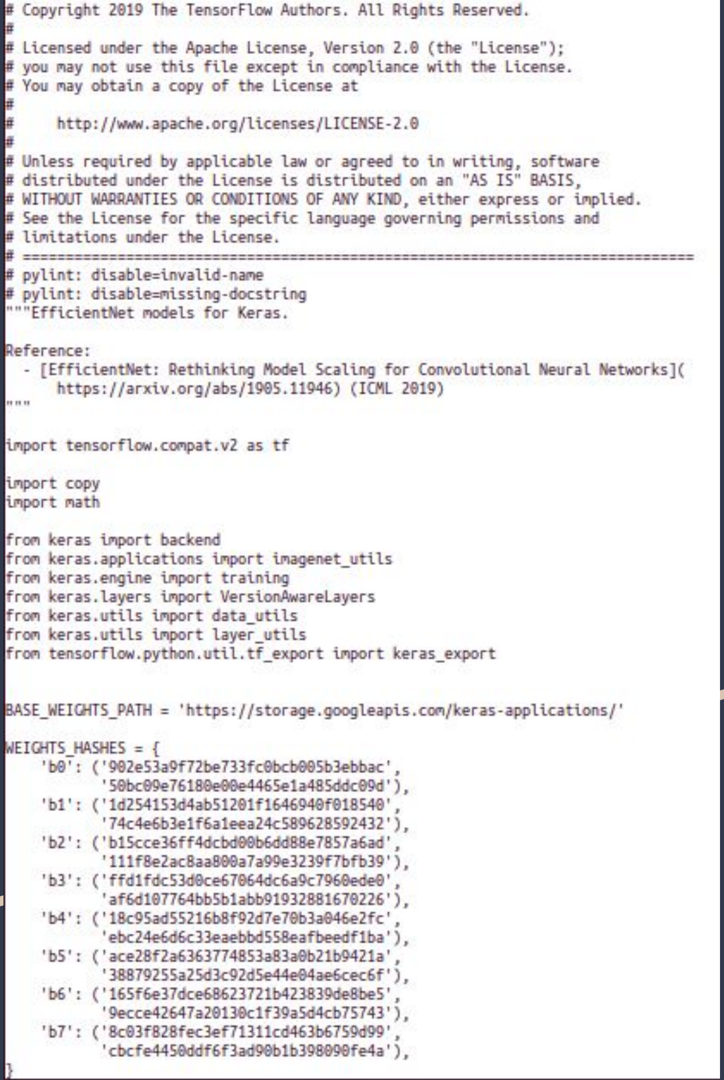


```
from django.test import TestCase

from .models import SlugPage


class RestrictedConditionsTests(TestCase):
    @classmethod
    def setUpTestData(cls):
        slugs = [
            'a',
            'a/a',
            'a/b',
            'a/b/a',
            'x',
            'x/y/z',
        ]
        SlugPage.objects.bulk_create([SlugPage(slug=slug) for slug in slugs])

    def test_restrictions_with_no_joining_columns(self):
        """
        It's possible to create a working related field that doesn't
        use any joining columns, as long as an extra restriction is supplied.
        """
        a = SlugPage.objects.get(slug='a')
        self.assertEqual(
            [p.slug for p in SlugPage.objects.filter(ascendants=a)],
            ['a', 'a/a', 'a/b', 'a/b/a'],
        )
        self.assertEqual(
            [p.slug for p in a.descendants.all()],
            ['a', 'a/a', 'a/b', 'a/b/a'],
        )

        aba = SlugPage.objects.get(slug='a/b/a')
        self.assertEqual(
            [p.slug for p in SlugPage.objects.filter(descendants__in=[aba])],
            ['a', 'a/b', 'a/b/a'],
        )
        self.assertEqual(
            [p.slug for p in aba.ascendants.all()],
            ['a', 'a/b', 'a/b/a'],
        )

    def test_empty_join_conditions(self):
        x = SlugPage.objects.get(slug='x')
        message = "Join generated an empty ON clause."
        with self.assertRaisesMessage(ValueError, message):
            list(SlugPage.objects.filter(containers=x))
```

BUG

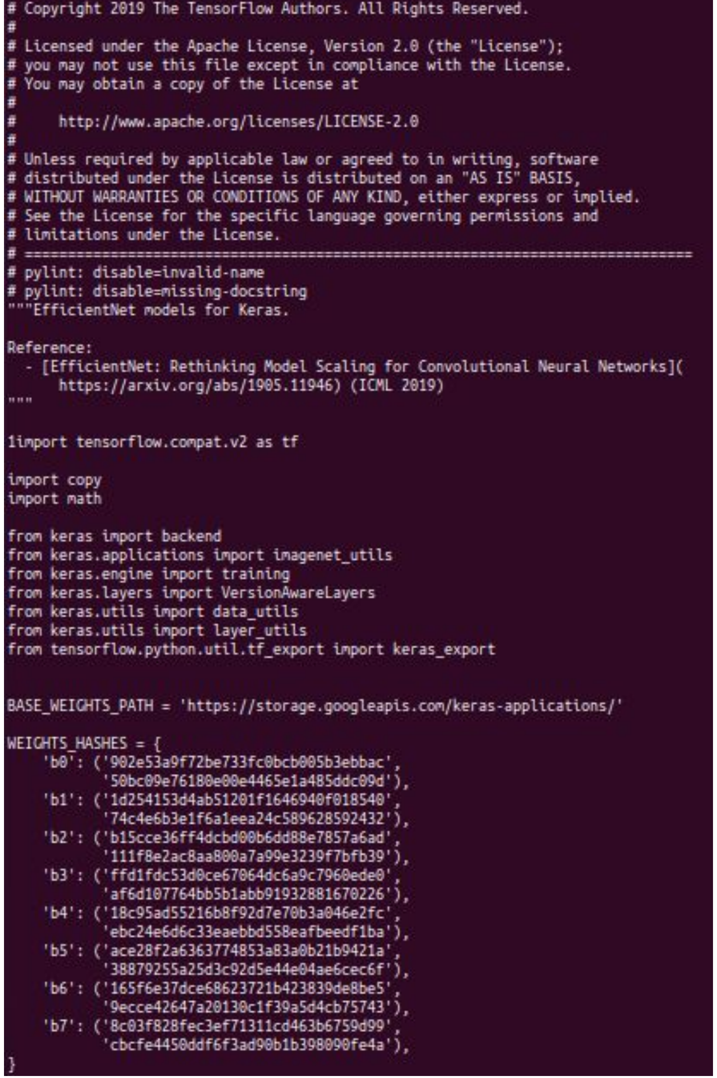


```
from django.test import TestCase

from .models import SlugPage


class RestrictedConditionsTests(TestCase):
    @classmethod
    def setUpTestData(cls):
        slugs = [
            'a',
            'a/a',
            'a/b',
            'a/b/a',
            'x',
            'x/y/z',
        ]
        SlugPage.objects.bulk_create([SlugPage(slug=slug) for slug in slugs])

    def test_restrictions_with_no_joining_columns(self):
        """
        It's possible to create a working related field that doesn't
        use any joining columns, as long as an extra restriction is supplied.
        """
        a = SlugPage.objects.get(slug='a')
        self.assertEqual(
            [p.slug for p in SlugPage.objects.filter(ascendants=a)],
            ['a', 'a/a', 'a/b', 'a/b/a'],
        )
2        self.assertEqual(
            [p.slug for p in a.descendants.all()],
            ['a', 'a/a', 'a/b', 'a/b/a'],
        )

        aba = SlugPage.objects.get(slug='a/b/a')
        self.assertEqual(
            [p.slug for p in SlugPage.objects.filter(descendants__in=[aba])],
            ['a', 'a/b', 'a/b/a'],
        )
        self.assertEqual(
            [p.slug for p in aba.ascendants.all()],
            ['a', 'a/b', 'a/b/a'],
        )

    def test_empty_join_conditions(self):
        x = SlugPage.objects.get(slug='x')
        message = "Join generated an empty ON clause."
        with self.assertRaisesMessage(ValueError, message):
            list(SlugPage.objects.filter(containers=x))
```



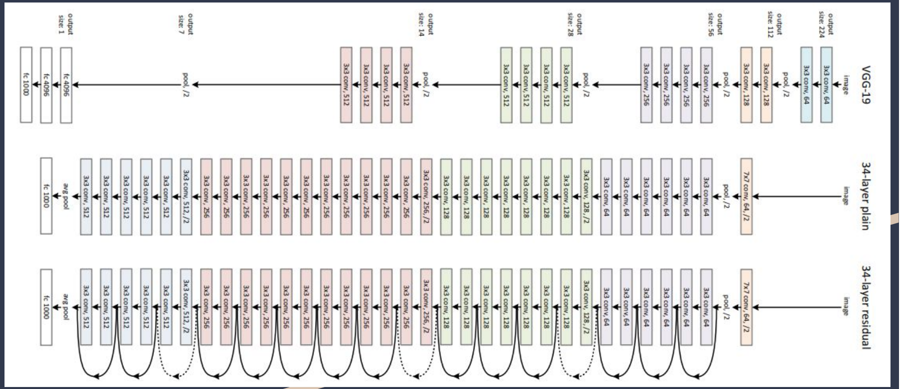
```
# Copyright 2019 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
# pylint: disable=invalid-name
# pylint: disable=missing-docstring
"""EfficientNet models for Keras.

Reference:
  - [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](
      https://arxiv.org/abs/1905.11946) (ICML 2019)
"""

import tensorflow.compat.v2 as tf

import copy
import math

from keras import backend
from keras.applications import imagenet_utils
from keras.engine import training
from keras.layers import VersionAwareLayers
from keras.utils import data_utils
from keras.utils import layer_utils
from tensorflow.python.util.tf_export import keras_export


BASE_WEIGHTS_PATH = 'https://storage.googleapis.com/keras-applications/'

WEIGHTS_HASHES = {
    'b0': ('902e53a9f72be733fc0bcb005b3ebbac',
           '50bc09e76180e00e4465e1a485ddc09d'),
    'b1': ('1d254153d4ab51201f1646940f018540',
           '74c4e6b3e1f6a1eea24c589628592432'),
    'b2': ('b15cce36ff4dcbd00b6dd88e7857a6ad',
           '111f8e2ac8aa800a7a99e3239f7bfb39'),
    'b3': ('ffd1fdc53d0ce67064dc6a9c7960ede0',
           'af6d107764bb5b1abb91932881670226'),
    'b4': ('18c95ad55216b8f92d7e70b3a046e2fc',
           'ebc24e6d6c33eaebbd558eafbeedf1ba'),
    'b5': ('ace28f2a6363774853a83a0b21b9421a',
           '38879255a25d3c92d5e44e04ae6cec6f'),
    'b6': ('165f6e37dce68623721b423839de8be5',
           '9ecce42647a20130c1f39a5d4cb75743'),
    'b7': ('8c03f828fec3ef71311cd463b6759d99',
           'cbcfe4450ddf6f3ad90b1b398090fe4a'),
}
```

BUG



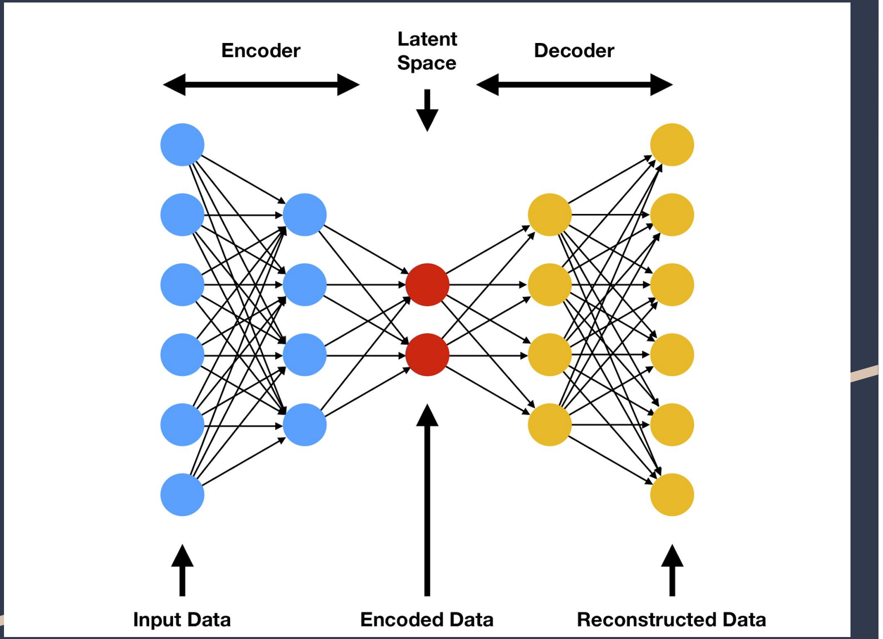
```
# Copyright 2019 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
# pylint: disable=invalid-name
# pylint: disable=missing-docstring
"""EfficientNet models for Keras.

Reference:
  - [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](
      https://arxiv.org/abs/1905.11946) (ICML 2019)
"""

1import tensorflow.compat.v2 as tf

import copy
import math

from keras import backend
from keras.applications import imagenet_utils
from keras.engine import training
from keras.layers import VersionAwareLayers
from keras.utils import data_utils
from keras.utils import layer_utils
from tensorflow.python.util.tf_export import keras_export


BASE_WEIGHTS_PATH = 'https://storage.googleapis.com/keras-applications/'

WEIGHTS_HASHES = {
    'b0': ('902e53a9f72be733fc0bcb005b3ebbac',
           '50bc09e76180e00e4465e1a485ddc09d'),
    'b1': ('1d254153d4ab51201f1646940f018540',
           '74c4e6b3e1f6a1eea24c589628592432'),
    'b2': ('b15cce36ff4dcbd00b6dd88e7857a6ad',
           '111f8e2ac8aa800a7a99e3239f7bfb39'),
    'b3': ('ffd1fdc53d0ce67064dc6a9c7960ede0',
           'af6d107764bb5b1abb91932881670226'),
    'b4': ('18c95ad55216b8f92d7e70b3a046e2fc',
           'ebc24e6d6c33eaebbd558eafbeedf1ba'),
    'b5': ('ace28f2a6363774853a83a0b21b9421a',
           '38879255a25d3c92d5e44e04ae6cec6f'),
    'b6': ('165f6e37dce68623721b423839de8be5',
           '9ecce42647a20130c1f39a5d4cb75743'),
    'b7': ('8c03f828fec3ef71311cd463b6759d99',
           'cbcfe4450ddf6f3ad90b1b398090fe4a'),
}
```

Transfer Learning and Fine Tuning



- Next attempt: ResNet150
- Much deeper pretrained network
- Chopped off last layer and modified for our needs
- Could not obtain accuracy > 50% either...

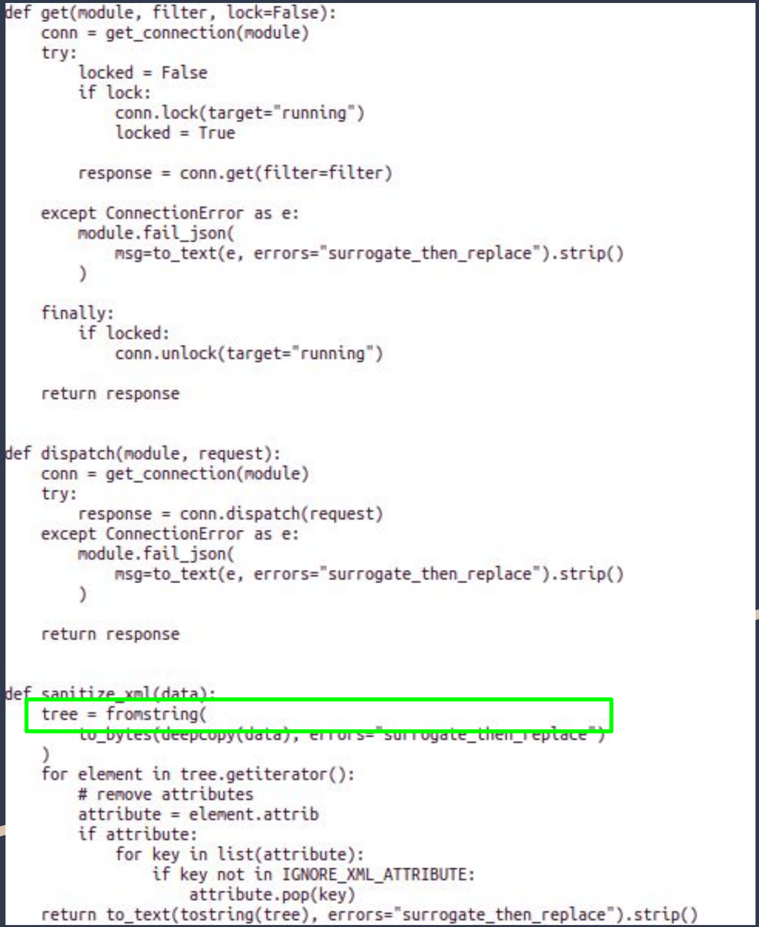
Anomaly Detection



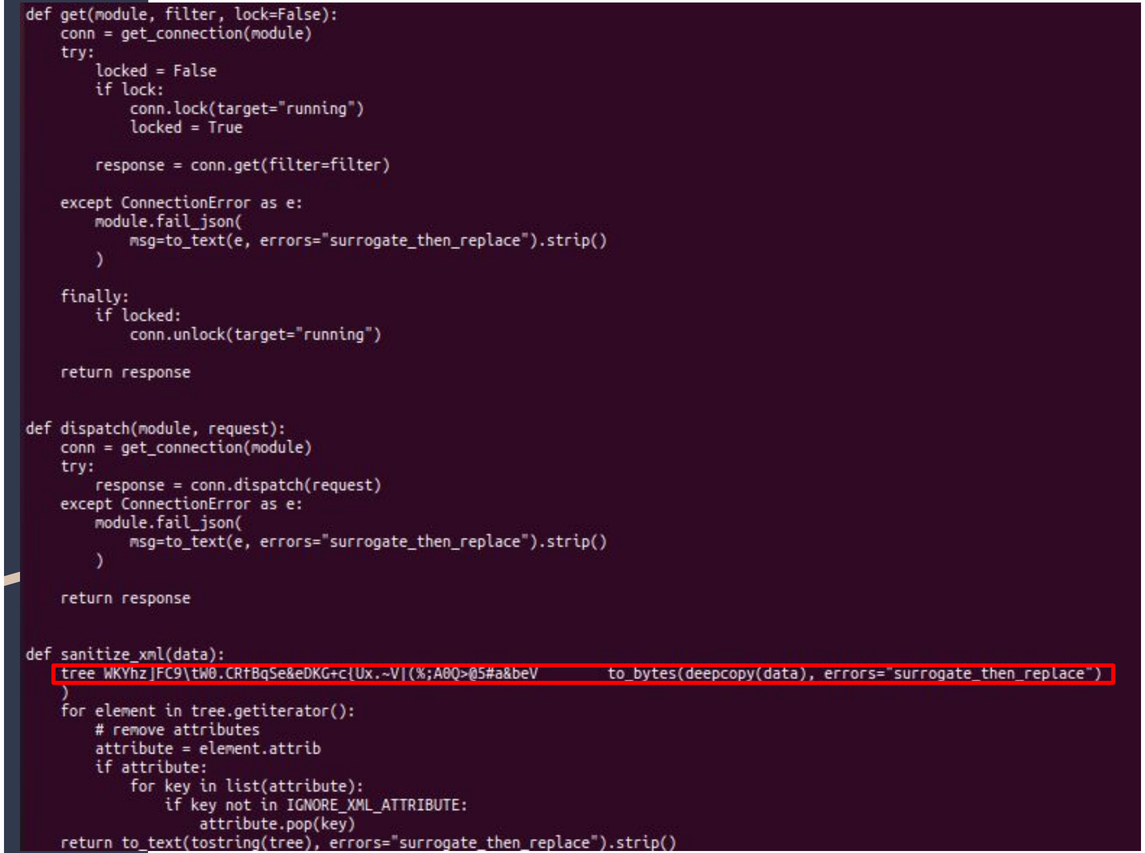
- Moved away from CNNs and looked into other ideas: Anomaly detection
- Had to research and learn about this (no prior experience, mainly worked with Vision DL systems)
- Autoencoder model could also not obtain accuracy > 50% either...

Dataset Revision

- One pixel/char change in a picture is tough to recognize if not aberration:
  - Imagine changing one pixel in a cat or dog pic for vision which is doing so well (would not change model prediction)
- Decided to see then how many consecutive chars/pixels need to be modified to get accuracy > 50%
  - Came ~45 chars (which is very high)



```
def get(module, filter, lock=False):
    conn = get_connection(module)
    try:
        locked = False
        if lock:
            conn.lock(target="running")
            locked = True

        response = conn.get(filter=filter)

    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    finally:
        if locked:
            conn.unlock(target="running")

    return response


def dispatch(module, request):
    conn = get_connection(module)
    try:
        response = conn.dispatch(request)
    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    return response


def sanitize_xml(data):
    tree = fromstring(
        to_bytes(deepcopy(data), errors="surrogate_then_replace")
    )
    for element in tree.getiterator():
        # remove attributes
        attribute = element.attrib
        if attribute:
            for key in list(attribute):
                if key not in IGNORE_XML_ATTRIBUTE:
                    attribute.pop(key)
    return to_text(tostring(tree), errors="surrogate_then_replace").strip()
```



```
def get(module, filter, lock=False):
    conn = get_connection(module)
    try:
        locked = False
        if lock:
            conn.lock(target="running")
            locked = True

        response = conn.get(filter=filter)

    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    finally:
        if locked:
            conn.unlock(target="running")

    return response


def dispatch(module, request):
    conn = get_connection(module)
    try:
        response = conn.dispatch(request)
    except ConnectionError as e:
        module.fail_json(
            msg=to_text(e, errors="surrogate_then_replace").strip()
        )

    return response


def sanitize_xml(data):
    tree WKYhzJFC9\tW0.CRfBqSe&eDKG+c{Ux.~V|(%;A0Q>@5#a&beV        to_bytes(deepcopy(data), errors="surrogate_then_replace")
    )
    for element in tree.getiterator():
        # remove attributes
        attribute = element.attrib
        if attribute:
            for key in list(attribute):
                if key not in IGNORE_XML_ATTRIBUTE:
                    attribute.pop(key)
    return to_text(tostring(tree), errors="surrogate_then_replace").strip()
```

Summary

Link to repo:
https://github.com/ShvetankPrakash/CodeErrorDetection

Findings

- If “any” bug allowed → acc > 90%
- If only change existing chars in code → 50%
- Need ~45 consecutive chars changed to get anything above 50%

Takeaways & What I learned

- Different from prior research I have done as an undergraduate, great lesson & preparation for future
- Anomaly detection & autoencoders

Future Work & Ideas

- Syntactic Bug vs Semantic Bug
    - Big challenge: how do you know code semantically incorrect without seeing full code and all its dependencies?
- Hyperparameter tuning if ever could find a different solution that works for detecting bugs in existing chars
- NLP model that generates source code and leverage it to detect errors in code?