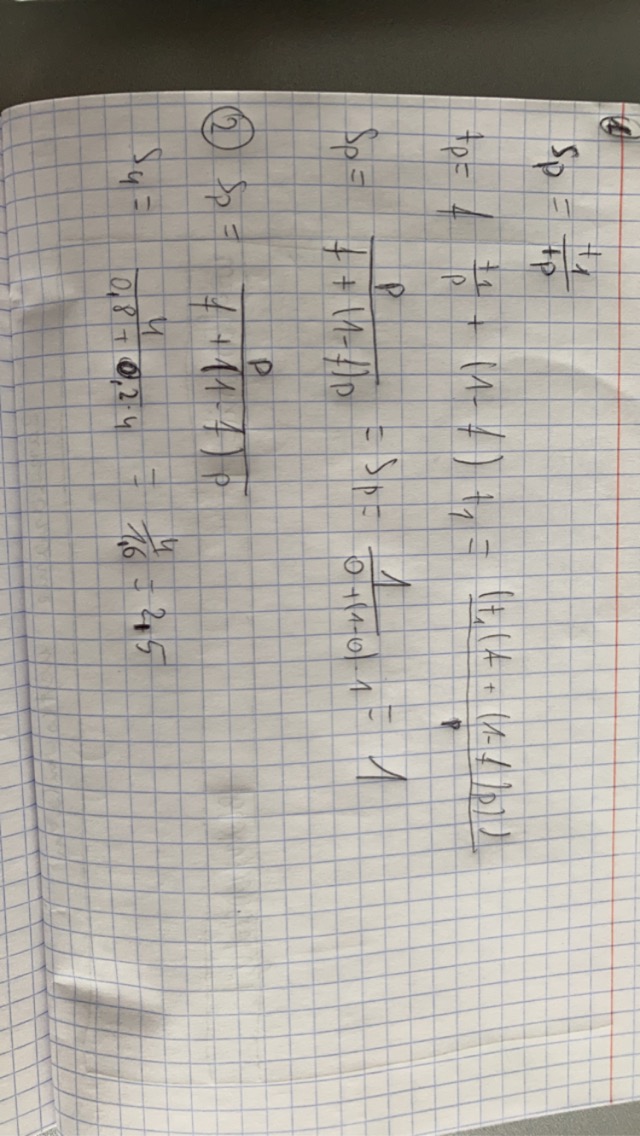
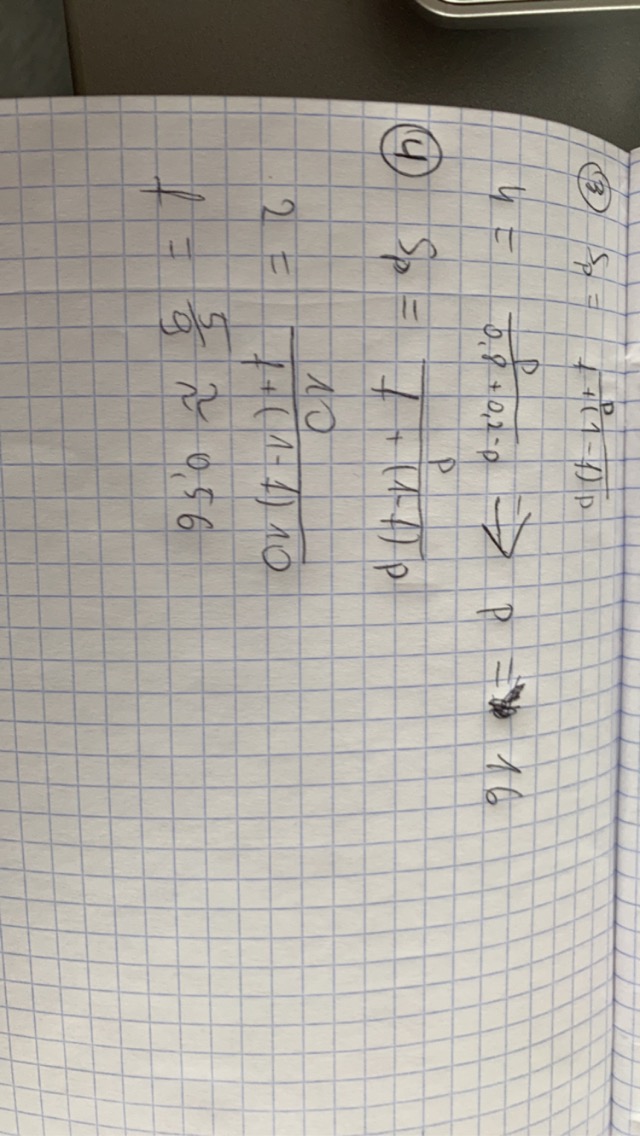
Zadanie 1 i 2.



Zadanie 3 i 4.



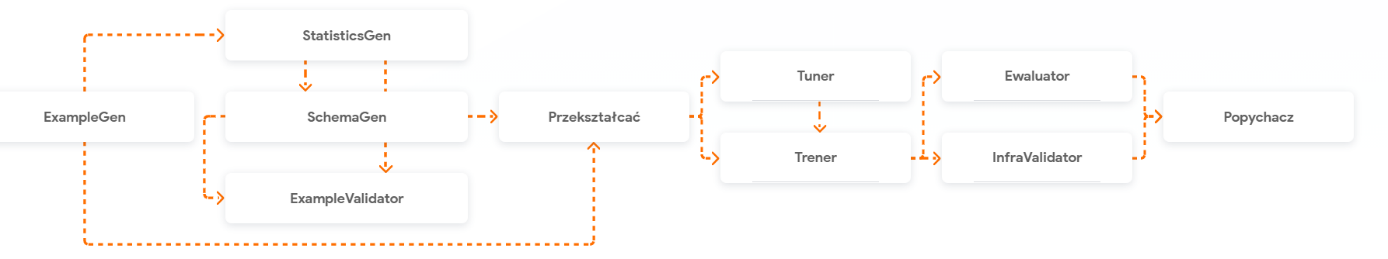
***Neural Structured Learning (NSL)*** to nowy paradygmat uczenia się do szkolenia sieci neuronowych poprzez wykorzystywanie sygnałów strukturalnych oprócz danych wejściowych. Struktura może być jawna, jak przedstawia wykres, lub niejawna jako wywołana zaburzeniami przeciwnymi.

Sygnały strukturalne są powszechnie używane do reprezentowania relacji lub podobieństw między próbkami, które mogą być oznaczone lub nie. Dlatego wykorzystanie tych sygnałów podczas uczenia sieci neuronowej wykorzystuje zarówno dane oznaczone, jak i nieoznaczone, co może poprawić dokładność modelu, szczególnie gdy ilość oznaczonych danych jest stosunkowo niewielka . Ponadto modele wytrenowane na podstawie próbek, które są generowane przez dodanie niekorzystnych perturbacji, okazały się odporne na złośliwe ataki , których celem jest wprowadzenie w błąd przewidywań lub klasyfikacji modelu.

Przykładowy kod:

import tensorflow as tf  
import neural\_structured\_learning as nsl  
  
# Prepare data.  
(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  
  
# Create a base model -- sequential, functional, or subclass.  
model = tf.keras.Sequential([  
    tf.keras.Input((28, 28), name='feature'),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(128, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])  
  
# Wrap the model with adversarial regularization.  
adv\_config = nsl.configs.make\_adv\_reg\_config(multiplier=0.2, adv\_step\_size=0.05)  
adv\_model = nsl.keras.AdversarialRegularization(model, adv\_config=adv\_config)  
  
# Compile, train, and evaluate.  
adv\_model.compile(optimizer='adam',  
                  loss='sparse\_categorical\_crossentropy',  
                  metrics=['accuracy'])  
adv\_model.fit({'feature': x\_train, 'label': y\_train}, batch\_size=32, epochs=5)  
adv\_model.evaluate({'feature': x\_test, 'label': y\_test})

***Potok TFX*** to sekwencja składników, które implementują potok ML, który jest specjalnie zaprojektowany do skalowalnych zadań uczenia maszynowego o wysokiej wydajności. Komponenty są budowane przy użyciu bibliotek TFX, które mogą być również używane indywidualnie.



***Neural style transfer (NTS)*** czyli manipulacja cyfrowymi obrazami lub filmami w celu dostosowania wyglądu lub stylu wizualnego innego obrazu. Algorytmy NST charakteryzują się wykorzystaniem głębokich sieci neuronowych w celu transformacji obrazu. Typowe zastosowania NST to tworzenie sztucznych dzieł sztuki z fotografii, na przykład przenoszenie wyglądu słynnych obrazów na fotografie dostarczone przez użytkownika.

Transfer stylu polega na wygenerowaniu obrazu o tej samej „treści” co obraz bazowy, ale z „stylem” innego obrazu (zazwyczaj artystycznego). Osiąga się to poprzez optymalizację funkcji straty, która składa się z 3 elementów: „utrata stylu”, „utrata treści” i „całkowita utrata zmienności”.

***Oblicz utratę transferu stylu:***

Najpierw musimy zdefiniować 4 funkcje użytkowe:

gram\_matrix (używane do obliczania utraty stylu)

style\_lossFunkcja, która utrzymuje Rodzony blisko lokalnych tekstur odniesienia obrazu stylu

content\_lossFunkcja, która prowadzi reprezentację wysokiego poziomu generowanego bliski obraz do obrazu bazowego

total\_variation\_lossFunkcja, utrata uregulowanie który utrzymuje generowanego obrazu lokalnie spójna

***TensorFlow Graphics*** ma na celu udostępnienie społeczności przydatnych funkcji graficznych poprzez zapewnienie zestawu zróżnicowanych warstw graficznych (np. Kamery, sploty siatki) oraz funkcje przeglądarki 3D (np. 3D TensorBoard), które można wykorzystać w modelach uczenia maszynowego wybór.

Przykładowy kod:

import numpy as np  
import tensorflow as tf  
import trimesh  
  
import tensorflow\_graphics.geometry.transformation as tfg\_transformation  
from tensorflow\_graphics.notebooks import threejs\_visualization  
  
# Download the mesh.  
!wget https://storage.googleapis.com/tensorflow-graphics/notebooks/index/cow.obj  
# Load the mesh.  
mesh = trimesh.load("cow.obj")  
mesh = {"vertices": mesh.vertices, "faces": mesh.faces}  
# Visualize the original mesh.  
threejs\_visualization.triangular\_mesh\_renderer(mesh, width=400, height=400)  
# Set the axis and angle parameters.  
axis = np.array((0., 1., 0.))  # y axis.  
angle = np.array((np.pi / 4.,))  # 45 degree angle.  
# Rotate the mesh.  
mesh["vertices"] = tfg\_transformation.axis\_angle.rotate(mesh["vertices"], axis,  
                                                        angle).numpy()  
# Visualize the rotated mesh.  
threejs\_visualization.triangular\_mesh\_renderer(mesh, width=400, height=400)

***TensorFlow SIG Addons*** to repozytorium wpisów społeczności, które są zgodne z ugruntowanymi wzorcami API, ale implementują nowe funkcje niedostępne w podstawowym TensorFlow.

TensorFlow natywnie obsługuje dużą liczbę operatorów, warstw, metryk, strat, optymalizatorów i nie tylko. Jednak w szybko zmieniającej się dziedzinie, takiej jak ML, istnieje wiele interesujących nowych rozwiązań, których nie można zintegrować z rdzeniem TensorFlow (ponieważ ich szerokie zastosowanie nie jest jeszcze jasne lub jest używane głównie przez mniejszą część społeczności).

Przykładowy kod:

import tensorflow as tf  
import tensorflow\_addons as tfa  
train,test = tf.keras.datasets.mnist.load\_data()  
x\_train, y\_train = train  
x\_train = x\_train[..., tf.newaxis] / 255.0  
  
# TFA layers and activations  
model = tf.keras.Sequential([  
  tf.keras.layers.Conv2D(filters=10, kernel\_size=(3,3),  
                         activation=tfa.activations.gelu),  
  tfa.layers.GroupNormalization(groups=5, axis=3),  
  tf.keras.layers.Flatten(),  
  tf.keras.layers.Dense(10, activation='softmax')  
])  
  
# TFA optimizers, losses and metrics  
model.compile(  
    optimizer=tfa.optimizers.RectifiedAdam(0.001),  
    loss=tfa.losses.TripletSemiHardLoss(),  
    metrics=[tfa.metrics.MultiLabelConfusionMatrix(num\_classes=10)])  
  
history = model.fit(x\_train, y\_train, epochs=10)