# Rozpoznávání druhu tkání

Tomáš Bíl, 500345

PA026

## 1 Úvod

V tomto projektu jsem se věnoval problematice klasifikace snímků tkání, které byly získány pomocí virtuální mikroskopie tkáně odebrané biopsií. Snímky jsou rozdéleny do tří kategorií podle druhu tkáné, ze které byl snímek pořízen. V datasetu jsou 3 třídy: kostní dřeň, prostata a prsní tuková tkáň.

## 2 Dataset

Dataset pochází ze dvou zdrojů (https://www.cancerimagingarchive.net/histopathology-imaging-on-tcia/, https://github.com/DeepPathology/MITOS_WSI_CMC). Obrázky jsou ve velmi detailním rozlišení. Všechny snímky pocházejí od pacientů s rakovinou.

| Četnosti | Prsní tkáň | Kostní dřeň | Prostata |
|---|---|---|---|
| Trénovací | 19450 | 4903 | 14194 |
| Testovací | 6519 | 1645 | 4685 |

## 3 Podobné projekty

Většina podobných projektu se zaměřuje na problémy podobné tomu mému, jako je klasifikace obrazu podle přítomnosti rakoviny, nebo klasifikace mutací v genech. Použití konvolučních neuronových je nejčastější řešení v rozpoznávání obrazu, bohužel není výpočetně možné zpracovávat celé snímky naráz kvůli jejich velikosti. Proto většina přístupů dělí snímky na menší dlaždice a ty potom zpracovává pomocí konvolučních sítí. Další přístup se, kterým jsem se setkal je reprezentace každé dlaždice pomocí pole jejích feature.

## 4 Teorie

Umělá neuronová síť je model velmi často využívaný v oboru informatiky, která se zabývá umělou inteligencí. Tento model má svou inspiraci v biologickém neuronu. Neurony jsou navzájem propojeny a každý neuron signál upraví a předá dál. Tímto způsobem se vstupní informace transformuje do stavu, který vyžadujeme. Zbývajícím problémem je, jak nastavit transformace, které vykonávají jednotlivé neurony, aby byla výsledná informace taková, jakou si ji představujeme. Způsob transformování signálu určují váhy. Tyto váhy jsou upravovány během fáze učení. Učení se rozlišuje na učení s učitelem a učení bez učitele. V případě učení s

učitelem jsou síti předkládány vstupy, neuronová síť vstup zpracuje a poté se výsledky ze sítě porovnají se správnými výsledky a podle toho se v síti upraví váhy, aby byly získané výsledky podobnější skutečným. Toto se opakuje vícekrát. Tento druh učení využívám i v této práci. Úprava vah se řídí algoritmem Backpropagation. Jednotlivé neurony bývají uspořádány do vrstev.

### 4.1 Konvoluční vrstva

Tato vrstva se využívá hlavně při rozpoznávání obrazu. Proto je tato vrstva jednou z hlavních součástí všech architektur, které jsou v této práci použity. Vstupem do této vrstvy je n-dimensionální pole, výstupem také. Prvky výstupu se nazývají feature mapy. Každá feature mapa je výsledkem aplikace konvolučního filtru na vstup. Počet filtrů definuje velikost výstupního pole. Velikost filtru udává, kolik vah se musí v rámci tohoto filtru natrénovat. Filtry mívají typicky souměrné velikosti dimenzí a na rozpoznání obrazu se zpravidla využívají dvourozměrné filtry. V praxi se osvědčily filtry s lichými velikostmi rozměrů, proto pokud se podíváme na aplikaci konvolučních vrstev, uvidíme hlavně filtry s velikostmi 3x3 a 5x5.

## 5 Zpracování dat

Snímky jsem rozdělil na dlaždice o velikosti 224x224. Z těchto dlaždic jsem potom vybral ty, které obsahují tkáň. Dále se preprocessing liší v závislosti na použitém modelu.

### 5.1 Konvoluční síť

Abych eliminoval rozdíly v zabarvení snímků z různých zdrojů, tak jsou všechny dlaždice převedeny na šedotónové a byla na nich provedena piecewise normalizace.

### 5.2 Dopředná neuronová síť

Každou dlaždici jsem převedl na pole feature délky 2048, které ji budou charakterizovat. Tento převod jsem provedl pomocí VGG50 předučené na ImageNet, výsledné pole je výstupem z average pooling vrstvy.

## 6 Modely

### 6.1 Konvoluční síť

Použitá konvoluční síť je tvořena 3 bloky složených z konvoluční a MaxPooling vrstvy. Po blocích následuje dropout vrstva a poté 2 dopředné vrstvy.

```
inp = layers.Input((256, 256, 3))

x = layers.Conv2D(16, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),) (inp)
x = layers.MaxPooling2D() (x)
x = layers.Conv2D(32, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),) (x)
x = layers.MaxPooling2D() (x)
x = layers.Conv2D(64, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),) (x)
x = layers.MaxPooling2D() (x)
x = layers.Dropout(0.2) (x)
x = layers.Flatten() (x)
x = layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),) (x)
out = layers.Dense(num_classes) (x)

model = Model(inp, out)

model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

## 6.2 Dopředná neuronová síť

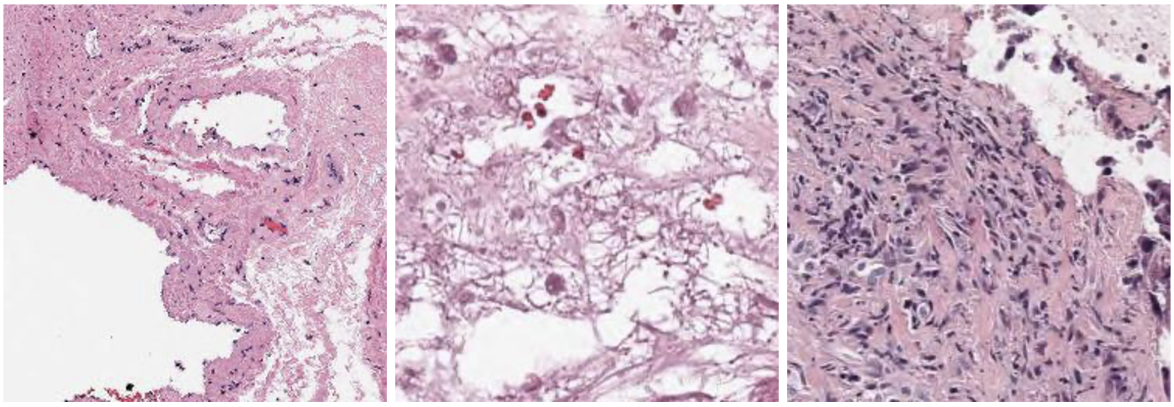
V tomto přístupu jsem použil síť se 3 dopřednými vrstvami o šířce 1024, 256 a 3.

```
def get_model_gene_prediction_sig(Ngenes):

    input_tile = Input(shape=(2048,))

    x = Dense(1024, activation='sigmoid')(input_tile)
    x = Dense(256, activation='sigmoid')(x)

    x_ts = Dense(
        Ngenes, activation='linear'
        )(x)

    model = Model(inputs=input_tile, outputs=x_ts)
    model.compile(optimizer=Adam(lr=0.001), loss='mean_squared_error', metrics=["accuracy"])

    return model
```

# 7 Výsledky

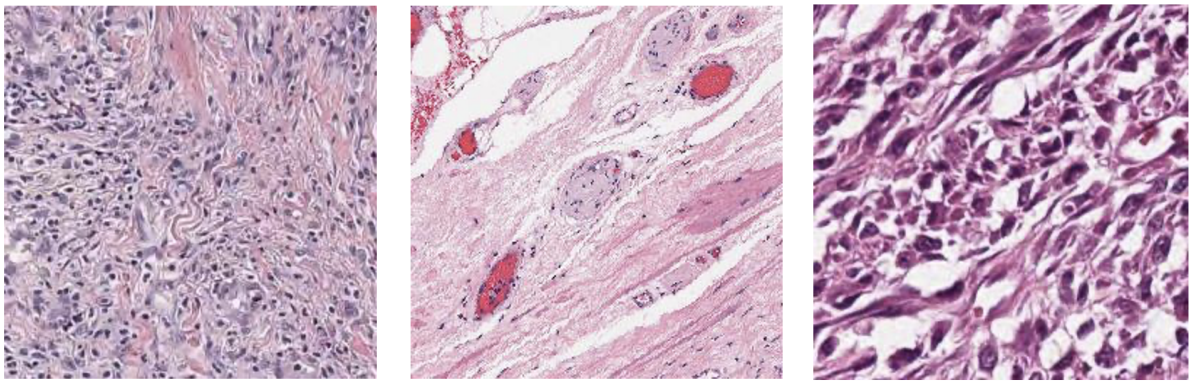
Oba přístupy dosáhly velmi dobrých výsledků.

| Výsledky | Přesnost | Loss |
|---|---|---|
| Dopředná neuronová síť | 0.995 | 0.003 |
| Konvoluční neuronová síť | 0.9789 | 0.1313 |

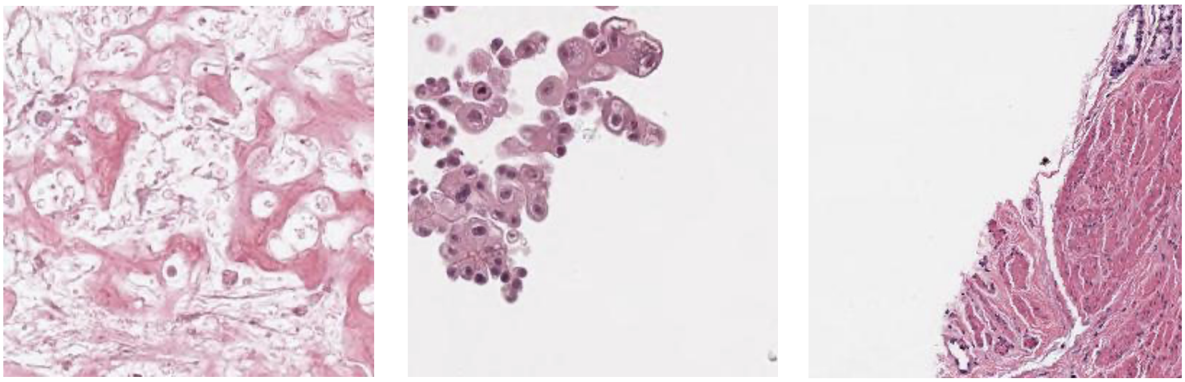


Správně klasifikované dlaždice konvoluční sítě

Špatně klasifikované dlaždice konvoluční sítě



Správně klasifikované dlaždice dopředné neuronové sítě



Špatně klasifikované dlaždice dopředné neuronové sítě

# 8 Screenshoty z trénování

```
Epoch 1/7
2021-05-27 08:15:35.939315: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudnn64_8.dll
2021-05-27 08:15:36.953890: I tensorflow/stream_executor/cuda/cuda_dnn.cc:367] Loaded cuDNN version 8101
964/964 [==============================] - 1071s 1s/step - loss: 1.0414 - accuracy: 0.7816 - val_loss: 0.3332 - val_accuracy: 0.9023
Epoch 2/7
964/964 [==============================] - 422s 438ms/step - loss: 0.2858 - accuracy: 0.9139 - val_loss: 0.1749 - val_accuracy: 0.9637
Epoch 3/7
964/964 [==============================] - 539s 559ms/step - loss: 0.2013 - accuracy: 0.9496 - val_loss: 0.1828 - val_accuracy: 0.9582
Epoch 4/7
964/964 [==============================] - 460s 477ms/step - loss: 0.1708 - accuracy: 0.9626 - val_loss: 0.1298 - val_accuracy: 0.9821
Epoch 5/7
964/964 [==============================] - 293s 304ms/step - loss: 0.1404 - accuracy: 0.9763 - val_loss: 0.1220 - val_accuracy: 0.9811
Epoch 6/7
964/964 [==============================] - 287s 297ms/step - loss: 0.1314 - accuracy: 0.9775 - val_loss: 0.1230 - val_accuracy: 0.9795
Epoch 7/7
964/964 [==============================] - 296s 307ms/step - loss: 0.1135 - accuracy: 0.9825 - val_loss: 0.1187 - val_accuracy: 0.9855
402/402 [==============================] - 266s 662ms/step - loss: 0.1313 - accuracy: 0.9789
test loss, test acc: [0.13133378326892853, 0.9789088368415833]
```

Trénování konvoluční sítě

```
Epoch 1/5
2021-05-27 09:26:20.884850: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublas64_11.dll
2021-05-27 09:26:21.247278: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cublasLt64_11.dll
1205/1205 [==============================] - 5s 4ms/step - loss: 0.0695 - accuracy: 0.9380
Epoch 2/5
1205/1205 [==============================] - 4s 4ms/step - loss: 0.0055 - accuracy: 0.9942
Epoch 3/5
1205/1205 [==============================] - 4s 4ms/step - loss: 0.0037 - accuracy: 0.9968
Epoch 4/5
1205/1205 [==============================] - 4s 3ms/step - loss: 0.0024 - accuracy: 0.9979
Epoch 5/5
1205/1205 [==============================] - 5s 4ms/step - loss: 0.0019 - accuracy: 0.9988
402/402 [==============================] - 1s 2ms/step - loss: 0.0042 - accuracy: 0.9949
test loss, test acc: [0.0041993409395217896, 0.999491234588623]
```

Trénování dopředné neuronové sítě

# 9 Instalace

Nainstalování dependencies pomocí "pip install -r requirements.txt".

# 10 Použití

"python model.py" spustí trénování konvoluční sítě.
"python model_msi" spustí trénování dopředné neuronové sítě.
"python preprocessing.py" spustí transformaci dlaždic na pole feature.
"python spatialization.py" vykreslí spatializaci dlaždice.
"python examples.py" a "python examples-msi.py" najdou špatně klasifikované dlaždice vypíše cestu k nim.