Slovenska Technicka Univerzita v Bratislave -
Fakulta elektrotechniky a informatiky

KONVOLUČNÁ NEURÓNOVÁ SIEŤ

Zadanie c.2

Autor: Vutianov Nikita

AIS ID: 112053

Rok: 2026

Obsah

Zadanie úlohy

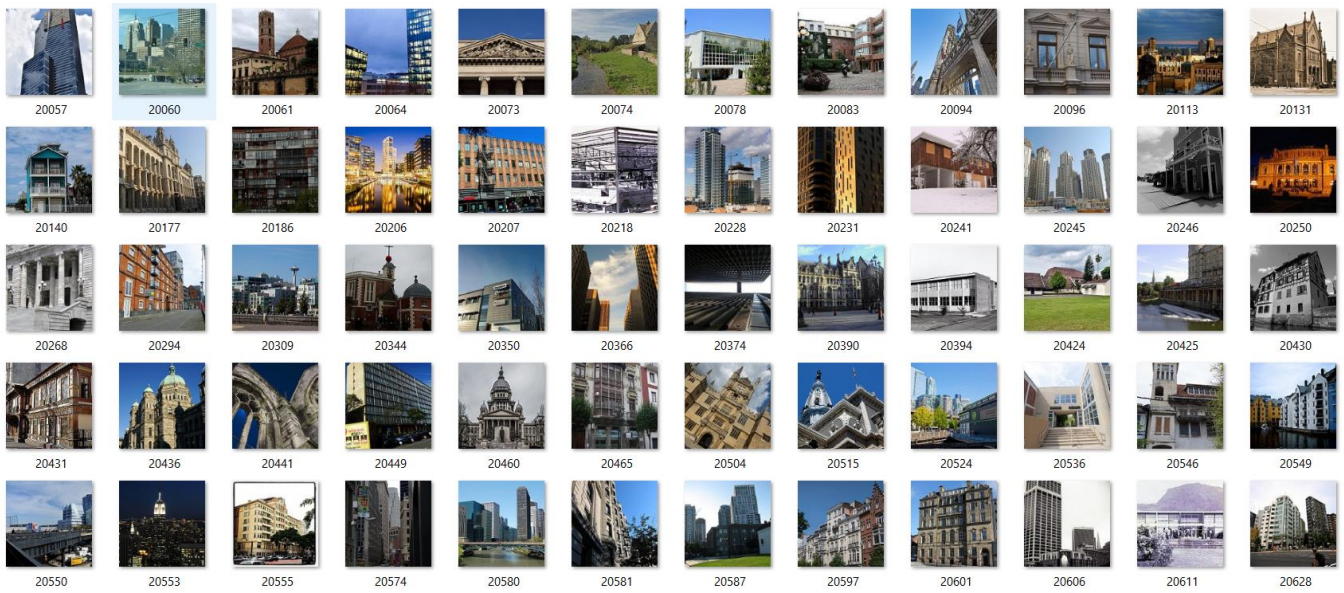
Cieľom zadania je vytvoriť konvolučnú neurónovú sieť na rozpoznávanie 4 kategórií obrázkov.
Najskôr pripravíme tréningové a testovacie dáta a navrhneme základnú architektúru siete. Následne vytvoríme viacero verzií siete s rôznymi parametrami (napr. počet filtrov, veľkosť filtrov, počet neurónov v Dense vrstve, batch_size a počet epoch). Každú verziu siete natrénujeme a vyhodnotíme jej presnosť na testovacej množine. Pre porovnanie výsledkov použijeme aj konfúznu maticu a klasifikačný report. Na záver uložíme modely a váhy a stručne zhodnotíme, ktorá verzia siete dosiahla najlepšie výsledky.

Postup riešenia

Dataset

Dataset – https://www.kaggle.com/datasets/puneet6060/intel-image-classification

Použitý dataset je rozdelený na tréningovú a testovaciu časť v štruktúre priečinkov **seg_train/seg_train/train** a **seg_test/seg_test/test**. Dataset obsahuje **4 triedy**: **buildings, forest, glacier, mountain**. Po načítaní dát cez flow_from_directory() bolo nájdených 9378 tréningových obrázkov a 1989 testovacích obrázkov, všetky rozdelené do 4 kategórií. Pre zjednotenie vstupu do CNN sme všetky obrázky zmenšili na rozmer 64×64 px a hodnoty pixelov sme normalizovali pomocou rescale = 1./255. Na tréningové dáta sme použili jednoduchú augmentáciu (rotácia, zoom a horizontálne prevrátenie) na zlepšenie generalizácie modelu.



Obr. 1: Ukážka datasetu – trieda buildings

Obr. 2: Ukážka obrázku každej tried

Na predspracovanie datasetu sme použili ImageDataGenerator z knižnice Keras. Všetky obrázky boli zmenšené na rozmer 64×64 px a normalizované pomocou rescale = 1./255. Pri tréningovej množine sme použili jednoduchú augmentáciu (rotácia, zoom a horizontálne prevrátenie). Pre načítanie dát sme použili funkciu flow_from_directory, pričom triedy sa automaticky načítali podľa názvov priečinkov. Parameter class_mode bol nastavený na categorical (viac tried). Nastavenie shuffle=False bolo ponechané podľa šablóny zadania.

Preprocessing tréningovej/testovacej množiny a kontrola tried

```python
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

training_set = train_datagen.flow_from_directory(
    r"C:\Users\aikid\Downloads\Zadanie2\archive (3)\seg_train\seg_train\train",
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale=1./255)

test_set = test_datagen.flow_from_directory(
    r"C:\Users\aikid\Downloads\Zadanie2\archive (3)\seg_test\seg_test\test",
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

print(training_set.samples)
```

```
Found 9378 images belonging to 4 classes.
Found 1989 images belonging to 4 classes.
9378
```

Po načítaní datasetu sa vypísal počet obrázkov v tréningovej a testovacej množine. Z výpisu vyplýva, že tréningová množina obsahuje **9378 obrázkov** a testovacia množina **1989 obrázkov**, pričom obe obsahujú **4 triedy**.

Trénovanie CNN

Na trenovanie využijeme neurónovú sieť z cvičenia 10 a video https://drive.google.com/file/d/1bDxQAemm89p5xxdntjSvGZyWc23JGUx5/view?usp=sharing.
Jej štruktúru môžeme vidieť nižšie. (je uvedena)

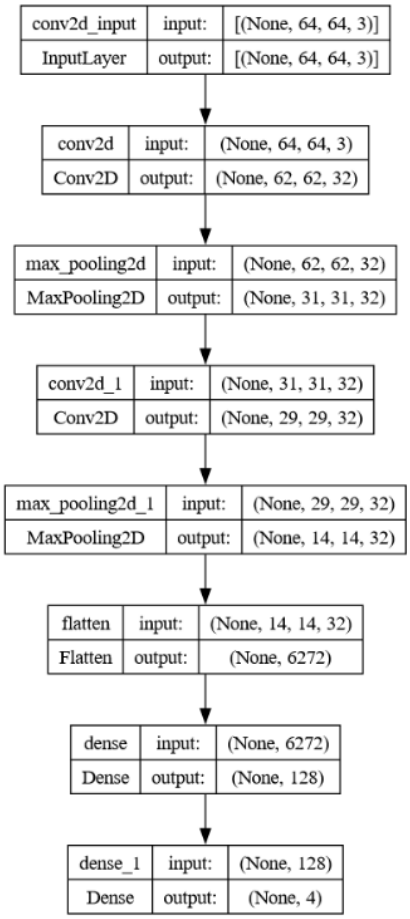
```python
# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - Convolution
cnn.add(tf.keras.layers.Conv2D(
    filters=32, kernel_size=3, activation='relu', input_shape=(64, 64, 3)
))

# Step 2 - Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))

# Adding a second pooling layer
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Adding a second convolutional layer
# cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))

# Adding a second pooling layer
# cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Step 5 - Output Layer
cnn.add(tf.keras.layers.Dense(units=4, activation='softmax'))
```

```python
# ===== Compiling the CNN =====
cnn.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 813,604
Trainable params: 813,604
Non-trainable params: 0
```

Obr.3: Siet a architektúra

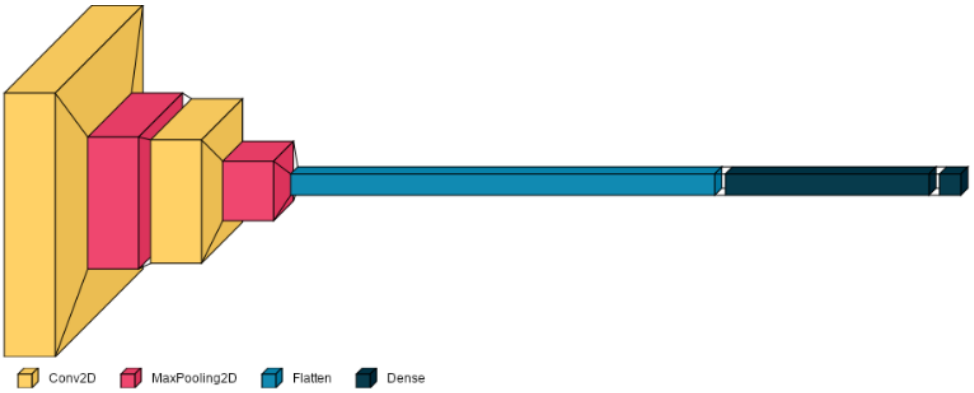
Plot architektury ktory bol ukazany z videa s pomocou from keras.utils.vis_utils import plot_model –



Obrazok vizualizacii siete

Vykreslíme pomocou kódu:

```
import visualkeras
from PIL import ImageFont

visualkeras.layered_view(cnn, legend=True)  # without custom font
font = ImageFont.truetype("arial.ttf", 12)
visualkeras.layered_view(cnn, legend=True, font=font, to_file='model_layered_cnn.png')
```

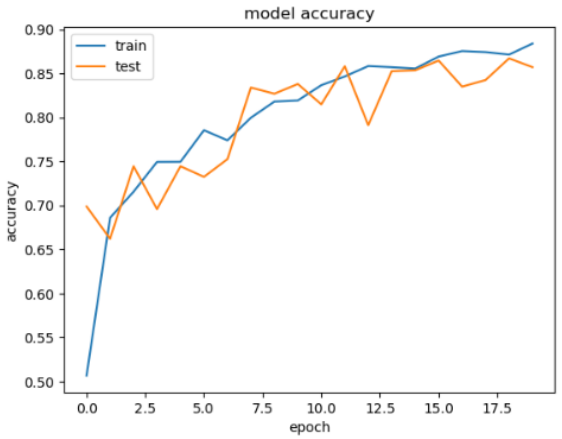


Obr. 6: Vizualizácia siete

Graf priebehu trénovania

Na vizualizáciu priebehu trénovania použijeme:

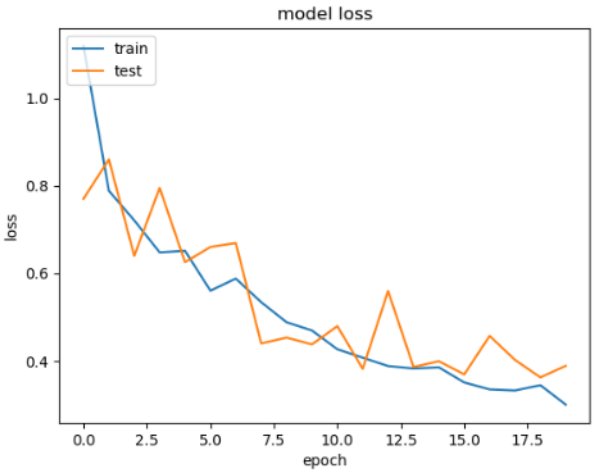
```
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')


plt.savefig('Model_accuracy_cnn.jpg', dpi=500)

plt.show()
```



Obr. 7: Graf priebehu trénovania accurancy

Na vizualizáciu loss funkcie použijeme:

```
import matplotlib.pyplot as plt

# summarize history for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')


plt.savefig('Model_loss_cnn.jpg', dpi=500)

plt.show()
```



Obr. 8: Graf priebehu trénovania los

Testovanie obrázku z každej triedy

Pre overenie funkčnosti natrénovanej siete sme otestovali klasifikáciu na samostatných obrázkoch. Testovací obrázok sa načíta zo súboru, zmenší sa na rozmer **64×64** a následne sa prevedie na pole. Keďže sieť očakáva vstup vo forme batchu, pridáme ešte jednu dimenziu pomocou np.expand_dims(). Výstupom siete je vektor pravdepodobností pre 4 triedy.

Výpis názvu predikovanej triedy

Sieť vracia pravdepodobnosti pre všetky triedy. Preto sa z výstupu vyberie index triedy s najvyššou hodnotou pomocou np.argmax(). Následne sa tento index premapuje na názov triedy cez slovník slovnik, ktorý zodpovedá poradiu tried v training_set.class_indices. Výsledok je vypísaný do konzoly vo forme: „Na obrázku je …“.

Obrázok s výpisom predikcie v obrázku

Okrem textového výpisu sme predikciu zobrazili aj priamo v obrázku. Na načítanie obrázka bol použitý OpenCV (cv2.imread). Predikovaný názov triedy sa vložil do obrázka pomocou cv2.putText(). Upravený obrázok sa následne uložil pod novým názvom (napr. glacier_with_signature.jpg) a zobrazil v Jupyter Notebooku.

```
import numpy as np
from keras.utils import load_img, img_to_array
img_path = r"C:/Users/aikid/Downloads/Zadanie2/glacier_example.jpg"

test_image = load_img(img_path, target_size=(64, 64))
test_image = img_to_array(test_image)
```

```
test_image = np.expand_dims(test_image, axis=0)
```

```
result = cnn.predict(test_image/255.0)
print("raw result:", result)
```

```
1/1 [==============================] - 0s 92ms/step
raw result: [[8.1914645e-03 4.1004198e-04 9.5851940e-01 3.2879081e-02]]
```

```
training_set.class_indices
```

```
{'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3}
```

```
slovnik = {0: "building", 1: "forest", 2: "glacier", 3: "mountain"}

predikcia = np.argmax(result, axis=1).astype(int)
predikcia
```

```
array([2])
```

```
print('Na obrazku je', slovnik[predikcia[0]])
```

```
Na obrazku je glacier
```

```
import cv2
im = cv2.imread(img_path, 1)

text = slovnik[predikcia[0]]
#font = cv2.FONT_HERSHEY_SIMPLEX
#cv2.putText(im, 'Christmas', (10,450), font, 3, (0, 255, 0), 2, cv2.LINE_AA)

cv2.putText(im, text, (20, 60), cv2.FONT_HERSHEY_SIMPLEX,  2, (0, 255, 0), 3, cv2.LINE_AA)
cv2.imwrite("glacier_with_signature.jpg", im)
print("Saved: glacier_with_signature.jpg")

from IPython.display import Image, display
display(Image(filename=img_path)
```

```
Saved: glacier_with_signature.jpg
```

```
from IPython.display import Image, display
display(Image(filename= r"C:/Users/aikid/Downloads/Zadanie2/glacier_with_signature.jpg" ))
```

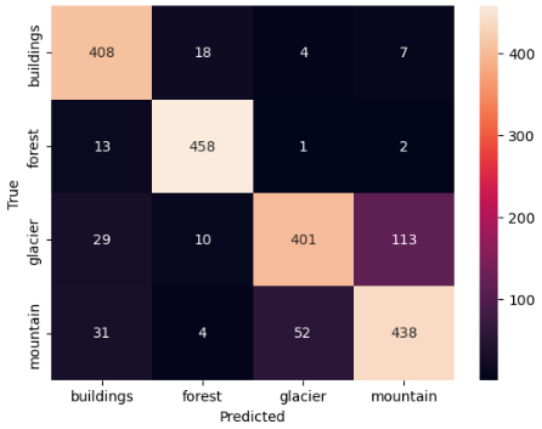
Obr. 9:Kod obrázok s s výpisom predikcie

Obr. 10: Predikované obrázky s textom a bez

Skóre presnosti, konfúzna matica, klasifikačný report

Presnosť siete sme vyhodnotili dvoma spôsobmi. Najskôr sme použili funkciu evaluate(), ktorá vrátila hodnoty loss = 0.39 a accuracy = 0.857 pre testovaciu množinu. Rovnaký výsledok presnosti sme následne overili aj pomocou accuracy_score(), kde sme dostali accuracy ≈ 0.857, čo potvrdzuje konzistentnosť výpočtu.
Konfúzna matica zobrazuje počet správnych a nesprávnych klasifikácií pre jednotlivé triedy (buildings, forest, glacier, mountain). Najvyššie hodnoty sa nachádzajú na diagonále, čo znamená, že model vo väčšine prípadov predikuje správne. Najlepšie je rozpoznávaná trieda forest (na diagonále 458 správnych prípadov) a veľmi dobré výsledky dosahuje aj buildings (408) a mountain (438). Slabším miestom je trieda glacier, kde sa objavuje výraznejšie zamieňanie s triedou mountain (z triedy glacier bolo 113 obrázkov predikovaných ako mountain). Tento jav je logický, pretože obe triedy môžu obsahovať podobné vizuálne prvky (sneh, skaly, horské pozadie), a preto je ich odlíšenie pre model náročnejšie.
Klasifikačný report ukazuje, že celková presnosť modelu je približne 86 % a hodnoty precision/recall/F1-score sú pre väčšinu tried vyrovnané. Najnižšie metriky sa objavujú pri triede mountain (precision = 0.78) a pri triede glacier (recall = 0.73), čo opäť zodpovedá pozorovanému zamieňaniu medzi týmito dvoma triedami.

```
63/63 [==============================] - 2s 37ms/step - loss: 0.3885 - accuracy: 0.8572
test loss, test acc: [0.3885203003883362, 0.8572146892547607]
63/63 [==============================] - 2s 38ms/step
accuracy_score: 0.8572146807440925
confusion matrix:
 [[408  18   4   7]
 [ 13 458   1   2]
 [ 29  10 401 113]
 [ 31   4  52 438]]
```




```
labels order: ['buildings', 'forest', 'glacier', 'mountain']
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       437
           1       0.93      0.97      0.95       474
           2       0.88      0.73      0.79       553
           3       0.78      0.83      0.81       525

    accuracy                           0.86      1989
   macro avg       0.86      0.86      0.86      1989
weighted avg       0.86      0.86      0.86      1989
```

Obr. 11: Úspešnosť predikcie

Uloženie, načitanie a skompilovanie siete

```python
from keras.models import save_model

neo_v1 = cnn
model_json = neo_v1.to_json()
with open("neo_v1.json", "w") as json_file:
    json_file.write(model_json)

save_model(neo_v1, "neo_v1_weights.hdf5")

with open("neo_v1.json", "r") as json_file:
    json_saved_model = json_file.read()

network_loaded = tf.keras.models.model_from_json(json_saved_model)


network_loaded.load_weights("neo_v1_weights.hdf5")


network_loaded.compile(
    loss="categorical_crossentropy",
    optimizer="Adam",
    metrics=["accuracy"]
)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896

max_pooling2d (MaxPooling2D  (None, 31, 31, 32)        0
)

conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248

max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)        0
2D)

flatten (Flatten)            (None, 6272)              0

dense (Dense)                (None, 128)               802944

dense_1 (Dense)              (None, 4)                 516

=================================================================
Total params: 813,604
Trainable params: 813,604
Non-trainable params: 0
_________________________________________________________________
```

Obr. 12: Znovu načitanie siete

Experimenty

V tejto časti som vytvoril 3 verzie konvolučnej neurónovej siete a porovnal ich správanie. Pri experimentoch som menil najmä počet filtrov, kernel_size, počet neurónov v Dense vrstve, ako aj regularizáciu pomocou Dropout. Cieľom bolo zlepšiť generalizáciu modelu a znížiť preučenie na tréningovej množine. Každý experiment má vlastnú architektúru a je trénovaný rovnakým spôsobom na rovnakom datasete. Každý experiment som realizoval v samostatnom Jupyter notebooku aby sa mi nemiešali výsledky a nastavenia. Pre každý experiment som použil vlastný názov súboru aj názov modelu, aby bolo jasné, ku ktorej architektúre patria grafy a výsledky. Modely a váhy som ukladal pod odlišnými názvami pre každý experiment.

Experiment 1

Zadanie2TestV1

V prvom experimente som upravil pôvodnú sieť a zameral som sa hlavne na regularizáciu a zmenu veľkosti filtrov. Do modelu som pridal Dropout vrstvy (0.25 po konvolučných blokoch a 0.50 pred výstupom), ktoré náhodne vypínajú časť neurónov počas tréningu. Týmto spôsobom sa sieť učí robustnejšie a menej sa naučí naspamäť tréningové dáta.
Zároveň som zmenil parametre druhej konvolučnej vrstvy: použil som 64 filtrov a upravil som kernel_size na 5×5, čo umožňuje zachytiť väčšie obrazové štruktúry. Na konci siete som použil Flatten() a Dense vrstvu s 256 neurónmi, aby mala sieť dostatočnú kapacitu na rozlíšenie 4 tried. Výstupná vrstva používa softmax pre multi-class klasifikáciu.
Priebeh učenia som sledoval pomocou grafov presnosti a straty pre tréningovú aj testovaciu množinu. Z grafu presnosti je vidieť stabilný rast výkonu: tréningová aj validačná presnosť postupne stúpajú a ku koncu tréningu sa ustália približne na úrovni ~0.86, pričom krivky sú si blízke – to naznačuje, že sieť sa nepreučuje výrazne a Dropout pomohol generalizácii. Graf straty zároveň ukazuje plynulý pokles pre obe množiny až na hodnotu približne ~0.38–0.40, bez veľkých výkyvov, čo znamená stabilné učenie.
Výsledné vyhodnotenie na testovacej množine som dosiahol pomocou evaluate() accuracy = 0.8587 a loss = 0.3844, čo potvrdzuje aj accuracy_score (0.8587). Konfúzna matica ukazuje, že najlepšie rozpoznávanou triedou je forest (456 správnych predikcií) a veľmi dobré výsledky sú aj pre buildings (396). Najväčšia chyba sa prejavila medzi triedami mountain a glacier – časť obrázkov mountain bola predikovaná ako glacier 90 prípadov a naopak glacier ako mountain 62 prípadov, čo je logické, keďže tieto triedy môžu obsahovať podobné vizuálne znaky (sneh, skaly, horské pozadie). Klasifikačný report potvrdzuje celkovú presnosť ~86 %, pričom trieda forest má najvyššie metriky (precision ~0.94, recall ~0.96) a slabším miestom ostáva najmä trieda mountain (nižší recall ~0.76).

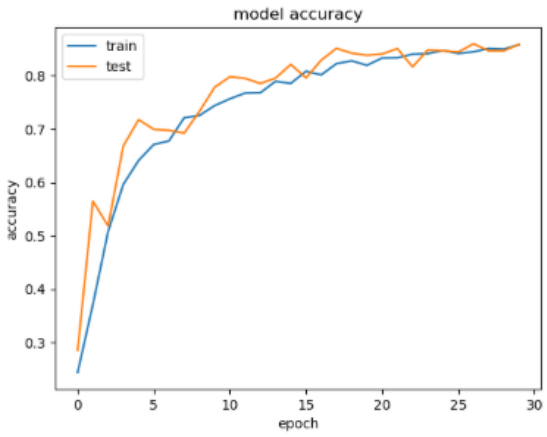
```python
# ===== Part 2 - Building the CNN =====
# Initialising the CNN
neo_v2 = tf.keras.models.Sequential()

# Conv32
neo_v2.add(tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', input_shape=(64, 64, 3)))
neo_v2.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
neo_v2.add(tf.keras.layers.Dropout(0.25))

# Conv64
neo_v2.add(tf.keras.layers.Conv2D(64, kernel_size=5, activation='relu'))## menil kernel_size=3 na 5
neo_v2.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
neo_v2.add(tf.keras.layers.Dropout(0.25))

# Flatten + Dense256
neo_v2.add(tf.keras.layers.Flatten())
neo_v2.add(tf.keras.layers.Dense(256, activation='relu'))
neo_v2.add(tf.keras.layers.Dropout(0.50))

# Output
neo_v2.add(tf.keras.layers.Dense(4, activation='softmax'))
```

```python
# ===== Training =====
hist = neo_v2.fit(x=training_set, validation_data=test_set, epochs=30)
# hist = neo_v2.fit(x=training_set, epochs=4)
```

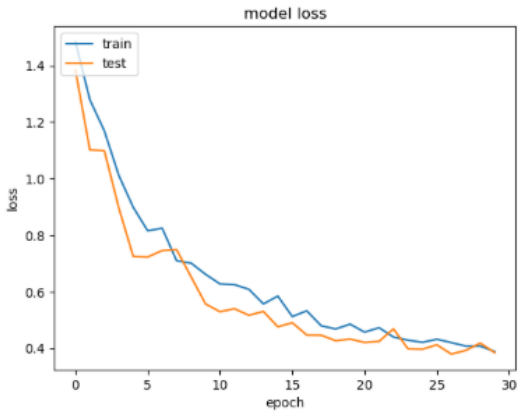
Obr. 13: Siet 1 - kód

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896

max_pooling2d (MaxPooling2D  (None, 31, 31, 32)        0
)

dropout (Dropout)            (None, 31, 31, 32)        0

conv2d_1 (Conv2D)            (None, 27, 27, 64)        51264

max_pooling2d_1 (MaxPooling  (None, 13, 13, 64)        0
2D)

dropout_1 (Dropout)          (None, 13, 13, 64)        0

flatten (Flatten)            (None, 10816)             0

dense (Dense)                (None, 256)               2769152

dropout_2 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 4)                 1028

=================================================================
Total params: 2,822,340
Trainable params: 2,822,340
Non-trainable params: 0
_________________________________________________________________
```

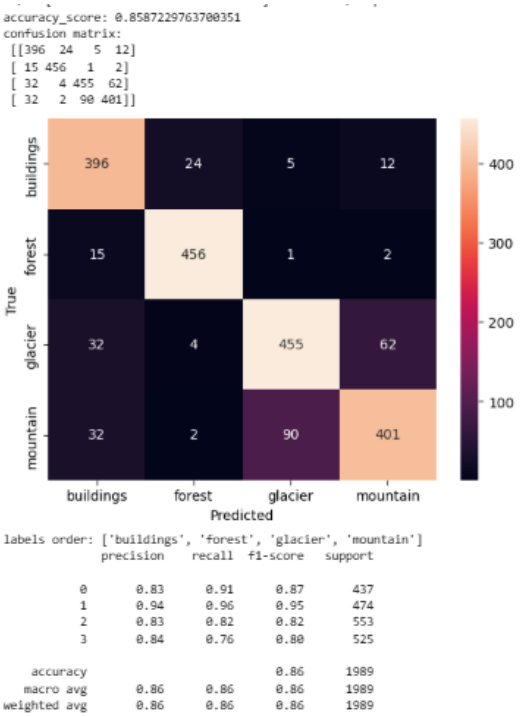
Obr. 14: Siet 1 - architektúra



Obr. 15: Siet 1 - accuracy



Obr. 16: Siet 1 - loss



Obr. 17: Siet 1 - Skóre presnosti, konfúzna matica, klasifikačný report

Experiment 2

Zadanie2TestV2

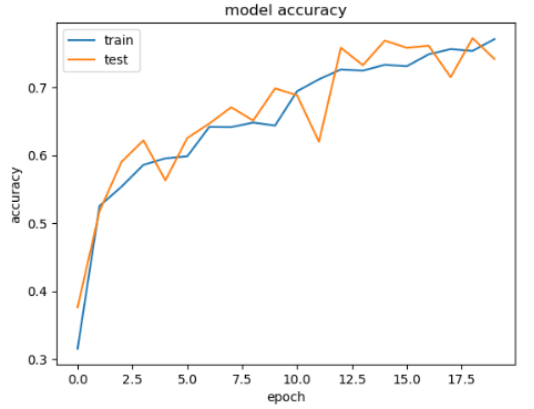
Cieľom bolo zvýšiť schopnosť siete učiť sa komplexnejšie vizuálne znaky a zároveň znížiť riziko preučenia. Architektúra začína konvolučnou vrstvou Conv2D(32) s väčším jadrom 5×5 a s padding='same', čo umožňuje zachytiť širšie obrazové štruktúry už na začiatku. Nasleduje MaxPooling 2×2, ktorý zmenší feature mapy a pomáha potláčať šum. Druhý blok používa Conv2D(64) s jadrom 3×3 (tiež s padding='same') a opäť MaxPooling, čím sa postupne zvyšuje kapacita siete a jej schopnosť rozlišovať jemnejšie detaily.
V ďalšej časti som pridal bottleneck vrstvu Conv2D(128) s jadrom 1×1, ktorá slúži na efektívnu kombináciu kanálov a preusporiadanie naučených príznakov bez veľkého nárastu výpočtovej náročnosti. Následne som použil ešte Conv2D(128) s jadrom 3×3, aby sieť dokázala z týchto bohatších reprezentácií extrahovať ešte detailnejšie vzor. Na rozdiel od verzií s Flatten som tu použil GlobalAveragePooling2D (GAP) – teda priemerovanie cez celú mapu príznakov. GAP výrazne znižuje počet parametrov oproti Flatten a často zlepšuje generalizáciu.
Po konvolučnej časti som použil Dense vrstvu s 256 neurónmi, ktorá vykonáva finálne rozhodovanie na základe extrahovaných príznakov. Pre zníženie preučenia som pridal Dropout(0.5), aby sa model nestal príliš závislý od konkrétnych kombinácií neurónov. Výstupná vrstva je Dense(4) so softmax. Významné je, že v tomto experimente som menil viacero parametrov oproti predošlým verziám: počet filtrov (32,64,128), veľkosť jadra (5×5 v prvej vrstve), pridaný bottleneck 1×1, typ redukcie (GAP namiesto Flatten), počet neurónov v Dense (256) a Dropout hodnota (0.5).
Z grafov accuracy je vidieť, že sieť sa učí postupne a stabilne – presnosť na tréningu aj na test/val rastie najmä v prvých epochách, neskôr sa krivky začnú približovať k ustálenej hodnote okolo ~0.75. Krivky tréningu a validácie sú relatívne blízko pri sebe, čo naznačuje, že model sa učí rozumne a nepreučuje sa extrémne. Graf loss zároveň ukazuje klesajúci trend – tréningový loss klesá plynulo a validačný loss tiež celkovo klesá, len s menšími lokálnymi zmenami, čo je bežné pri klasifikácii obrazov a pri menšom rozlíšení vstupu (64×64).
Na testovacej množine som dosiahol celkovú presnosť približne 0.74. Konfúzna matica ukazuje, že trieda forest je rozpoznávaná veľmi dobre (veľa správnych predikcií na diagonále), podobne aj buildings má slušnú úspešnosť. Najväčší problém však vzniká pri triedach glacier a mountain. Klasifikačný report tento jav potvrdzuje – metriky precision/recall sú zvyčajne stabilnejšie pre forest a buildings, zatiaľ čo pri dvojici glacier/mountain sú hodnoty slabšie kvôli častejšej zámene.

```python
# Initialising the CNN
neo_v3 = tf.keras.models.Sequential()


neo_v3.add(tf.keras.layers.Conv2D(32, kernel_size=5, activation='relu', input_shape=(64, 64, 3), padding='same'))
neo_v3.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Block 2 (64 фильтра, 3x3)
neo_v3.add(tf.keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'))
neo_v3.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Block 3 (1x1 bottleneck -> делает сеть другой)
neo_v3.add(tf.keras.layers.Conv2D(128, kernel_size=1, activation='relu', padding='same'))

# Block 4 (ещё один conv для "глубины")
neo_v3.add(tf.keras.layers.Conv2D(128, kernel_size=3, activation='relu', padding='same'))

# Вместо Flatten:
neo_v3.add(tf.keras.layers.GlobalAveragePooling2D())

# Dense часть (другое число нейронов)
neo_v3.add(tf.keras.layers.Dense(256, activation='relu'))
neo_v3.add(tf.keras.layers.Dropout(0.5))

# Output (4 triedy) - ТОЛЬКО ОДИН!
neo_v3.add(tf.keras.layers.Dense(4, activation='softmax'))
```

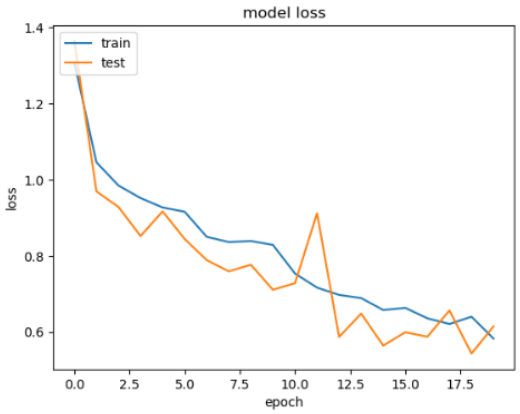
Obr. 18: Siet 2 - kód

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 64, 64, 32)        2432

 max_pooling2d_4 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       8320

 conv2d_8 (Conv2D)           (None, 16, 16, 128)       147584

 global_average_pooling2d_1   (None, 128)              0
 (GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 256)               33024

 dropout_4 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 4)                 1028

=================================================================
Total params: 210,884
Trainable params: 210,884
Non-trainable params: 0
_________________________________________________________________
```

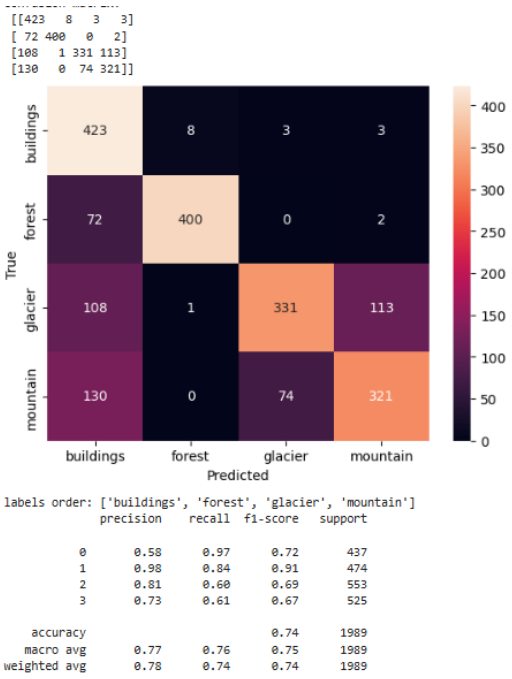
Obr. 19: Siet 2 - architektúra



Obr. 20: Siet 2 - accuracy



Obr. 21: Siet 2 - loss



Obr. 22: Siet 2 - Skóre presnosti, konfúzna matica, klasifikačný report

Experiment 3

Zadanie2TestV3

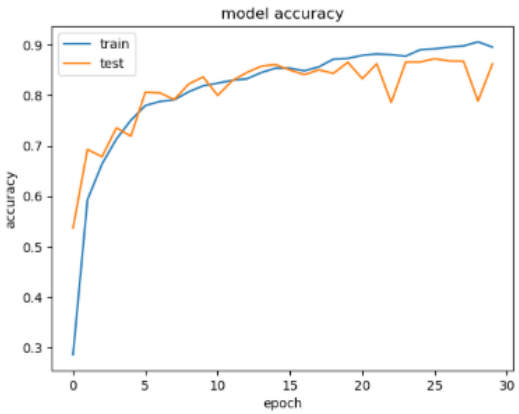
V tomto experimente som vytvoril hlbšiu sieť s tromi konvolučnými vrstvami, aby model dokázal extrahovať komplexnejšie príznaky z obrázkov. Architektúra začína vrstvou Conv2D s 32 filtrami a väčším jadrom 5×5, čo umožňuje zachytiť širšie obrazové štruktúry už v prvej fáze spracovania. Nasledujúca konvolučná vrstva používa 64 filtrov s kernel 3×3 a ďalšia vrstva 128 filtrov s kernel 3×3, čím sa postupne zvyšuje kapacita siete a schopnosť rozlišovať jemné detaily. Medzi konvolučnými vrstvami je použitý MaxPooling (2×2), ktorý znižuje rozmer feature máp a zároveň pomáha potláčať šum.
Po konvolučnej časti som použil Flatten, následne Dense vrstvu s 256 neurónmi, ktorá slúži na finálne rozhodovanie na základe extrahovaných príznakov. Pre zníženie preučenia som pridal Dropout(0.3). Výstupná vrstva je Dense(4) so softmax, keďže ide o klasifikáciu do 4 tried.
Z grafov accuracy/loss je vidieť stabilné učenie: tréningová aj validačná presnosť rastú rýchlo v prvých epochách a ku koncu sa ustália približne na úrovni ~0.86–0.90. Krivky tréningu a validácie sú si blízke, čo naznačuje relatívne dobrú generalizáciu. Strata (loss) postupne klesá, pričom validačný loss má mierne výkyvy v neskorších epochách, čo je bežné pri menšom datasete a naznačuje, že model už je blízko maxima výkonu.
Celková presnosť na testovacej množine dosahuje približne 0.86–0.87. Konfúzna matica ukazuje, že triedy buildings a forest sú rozpoznávané veľmi dobre. Najväčšie zámenné chyby vznikajú medzi triedami glacier a mountain.

```python
# Initialising the CNN
neo_v4 = tf.keras.models.Sequential()

# Conv1:
neo_v4.add(tf.keras.layers.Conv2D(filters=32, kernel_size=5, activation='relu', input_shape=(64, 64, 3)))
neo_v4.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Conv2:
neo_v4.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
neo_v4.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Conv3:
neo_v4.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, activation='relu'))
neo_v4.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))


neo_v4.add(tf.keras.layers.Flatten())
neo_v4.add(tf.keras.layers.Dense(units=256, activation='relu'))
neo_v4.add(tf.keras.layers.Dropout(0.3))

# Output
neo_v4.add(tf.keras.layers.Dense(units=4, activation='softmax'))
```

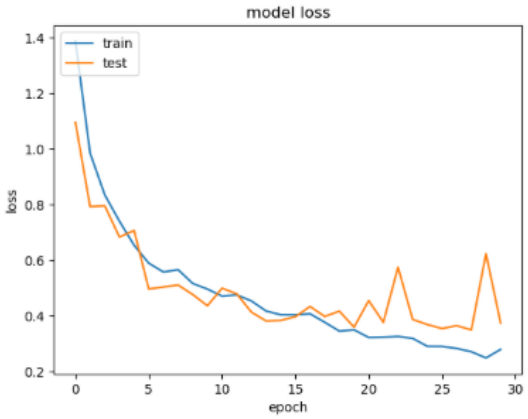
Obr. 23: Siet 3 - kód

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 60, 60, 32)        2432

max_pooling2d_7 (MaxPooling  (None, 30, 30, 32)        0
2D)

conv2d_8 (Conv2D)            (None, 28, 28, 64)        18496

max_pooling2d_8 (MaxPooling  (None, 14, 14, 64)        0
2D)

conv2d_9 (Conv2D)            (None, 12, 12, 128)       73856

max_pooling2d_9 (MaxPooling  (None, 6, 6, 128)         0
2D)

flatten_3 (Flatten)          (None, 4608)              0

dense_6 (Dense)              (None, 256)               1179904

dropout_6 (Dropout)          (None, 256)               0

dense_7 (Dense)              (None, 4)                 1028

=================================================================
Total params: 1,275,716
Trainable params: 1,275,716
Non-trainable params: 0
_________________________________________________________________
```

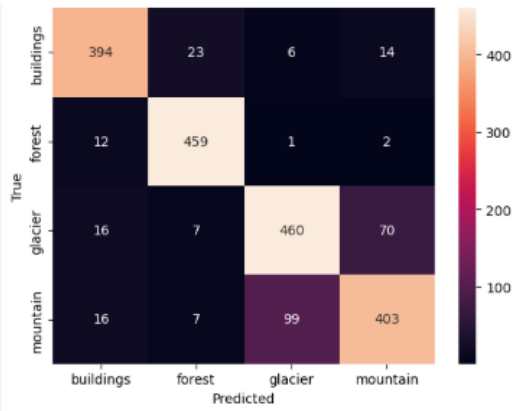
Obr. 24: Siet 3 - architektúra



Obr. 25: Siet 3 - accuracy



Obr. 26: Siet 3 - loss



labels order: ['buildings', 'forest', 'glacier', 'mountain']

	precision	recall	f1-score	support
0	0.90	0.90	0.90	437
1	0.93	0.97	0.95	474
2	0.81	0.83	0.82	553
3	0.82	0.77	0.79	525
accuracy			0.86	1989
macro avg	0.87	0.87	0.87	1989
weighted avg	0.86	0.86	0.86	1989

Obr. 27: Siet 3 - Skóre presnosti, konfúzna matica, klasifikačný report

Vyhodnotenie experimentov

Najlepšie výsledky dosiahol Experiment 3 (3×Conv + Flatten + Dense256 + Dropout 0.3), kde bola testovacia presnosť približne 0.86–0.87 a triedy buildings a forest boli rozpoznávané veľmi spoľahlivo. Druhý najlepší bol Experiment 1 (2×Conv Dropout Dense256) s presnosťou okolo ~0.86, pričom hlavné chyby vznikali najmä medzi glacier a mountain. Najhoršie dopadol Experiment 2 (bottleneck 1×1 + GAP), ktorý dosiahol približne ~0.74 a častejšie zamieňal najmä triedy glacier a mountain.

Experiment 1

Výsledná úspešnosť prvého experimentu bola približne 86 % (evaluate/accuracy_score ~0.8587). Z grafov priebehu tréningu vidíme stabilný rast presnosti a plynulý pokles loss pre tréning aj test, pričom krivky sú si pomerne blízke – to naznačuje, že model sa nepreučuje výrazne a Dropout pomohol generalizácii. Pri pohľade na konfúznu maticu vidíme, že sieť veľmi dobre klasifikuje triedy forest a buildings, no najväčšie chyby vznikajú pri dvojici glacier - mountain, kde sa triedy často zamieňajú kvôli podobným vizuálnym prvkom (sneh, skaly, horské pozadie). Celkovo bol tento experiment veľmi úspešný a priniesol stabilné a vyvážené výsledky.
neo_v2 - bola trénovaná 35-40 minút

Experiment 2

V druhom experimente som použil architektúru s bottleneck vrstvou 1×1 a GlobalAveragePooling2D, čo výrazne znížilo počet parametrov a malo zlepšiť generalizáciu. Výsledná presnosť na testovacej množine však dosiahla len približne 74 %, čo je citeľne menej ako pri experimente 1 a 3. Z grafov tréningu je vidieť, že učenie síce postupovalo, ale model sa zastavil na nižšej úrovni presnosti a výstup bol menej stabilný. Konfúzna matica ukazuje výraznejšie chyby najmä medzi triedami glacier a mountain, pričom nesprávne predikcie boli častejšie aj oproti ostatným triedam. Tento experiment ukázal, že GAP prístup je síce efektívny, ale v našom prípade nedokázal dosiahnuť takú presnosť ako klasické riešenie s Flatten. neo_v3 - bola trénovaná 30-33 minút

Experiment 3

Tretí experiment bol zameraný na vytvorenie hlbšej siete s tromi konvolučnými vrstvami (32,64,128) a s väčším jadrom 5×5 v prvej vrstve, aby model zachytil širšie štruktúry už na začiatku. Výsledná úspešnosť na testovacej množine bola približne 87 %, čo je najlepší výsledok spomedzi experimentov. Z grafov accuracy/loss je vidieť stabilné učenie – presnosť tréningu aj testu rastie a ku koncu sa drží vysoko, pričom loss klesá bez výrazných výkyvov. Konfúzna matica ukazuje veľmi dobré výsledky pre triedy buildings a forest, a zároveň zlepšenie oproti slabšiemu modelu, aj keď stále zostáva najväčší problém v zámene medzi glacier a mountain. Celkovo tento experiment vyšiel najlepšie a ukázal, že hlbšia sieť s vyšším počtom filtrov má najlepší výkon pre náš dataset.
neo_v3 - bola trénovaná 28-33 minút

Záver

V tejto úlohe som vytvoril a otestoval tri rôzne CNN architektúry na klasifikáciu 4 kategórií obrázkov. Postupne som menil parametre siete a výsledky som vyhodnotil pomocou grafov accuracy/loss, evaluate, konfúznej matice a klasifikačného reportu. Najlepší výsledok mi dal tretí experiment (neo_v4), ktorý dosiahol približne 86 %(0.86) presnosť na testovacích dátach. Ukázalo sa, že najťažšie je rozlišovať triedy glacier a mountain, pretože majú podobné vizuálne črty. Aj napriek tomu sa mi podarilo zostrojiť sieť, ktorá dokáže triedy rozpoznávať spoľahlivo a výsledky sú jasne porovnateľné medzi experimentmi.