Komplexní výuková příručka

Principy a implementace automatizované detekce a klasifikace mincí pomocí počítačového vidění a hlubokého učení

Úvod

Vítejte v této rozsáhlé výukové příručce, která je koncipována jako vyčerpávající průvodce světem automatizované analýzy obrazu se zaměřením na specifickou a pedagogicky vděčnou úlohu: detekci a rozpoznávání mincí. Tato úloha v sobě unikátním způsobem snoubí dvě fundamentální disciplíny moderní umělé inteligence – klasické počítačové vidění (Computer Vision), zaměřené na geometrickou analýzu a zpracování signálu, a hluboké učení (Deep Learning), které přináší schopnost sémantické klasifikace a generalizace.

Cílem tohoto textu není pouze poskytnout funkční kód, ale především vybudovat hluboké teoretické porozumění principům, které umožňují strojům "vidět" a "chápat" vizuální data.

Numismatika a automatizované třídění mincí představují pro počítačové vidění zajímavou výzvu. Mince jsou objekty s pevně danou geometrií (kruhový tvar), což umožňuje využití robustních matematických transformací, avšak jejich povrch je vysoce variabilní. Různé stupně opotřebení, oxidace (patina), odlesky kovu, variabilní osvětlení a jemné detaily ražby činí z klasifikace netriviální problém, který nelze řešit pouhým porovnáváním pixelů.1

Historicky vyžadovala identifikace mincí expertní znalosti a manuální kontrolu, což bylo časově náročné a subjektivní. S příchodem konvolučních neuronových sítí (CNN) a pokročilých algoritmů segmentace se však otevírají možnosti pro plnou automatizaci, která nachází uplatnění od prodejních automatů až po archeologický výzkum a mobilní aplikace pro sběratele.1

Tato příručka je strukturována do logických bloků, které kopírují reálný tok dat v aplikaci počítačového vidění: od pořízení obrazu, přes jeho předzpracování a segmentaci, až po extrakci příznaků a finální klasifikaci pomocí neuronových sítí. U každého tématu budeme důsledně rozlišovat mezi teoretickým principem (matematickým a algoritmickým pozadím) a praktickou implementací (kódem v Pythonu s využitím knihoven OpenCV a PyTorch/TensorFlow).

Důraz bude kladen na pochopení kauzálních souvislostí – proč volíme určité parametry, jaký dopad má šum na detektory hran a jak interpretovat chování neuronové sítě během tréninku.

Kapitola 1: Digitální reprezentace obrazu a předzpracování dat

Než se pustíme do detekce mincí, musíme pochopit, s jakými daty pracujeme. Digitální obraz, ačkoliv se lidskému oku jeví jako spojitá scéna, je pro počítač diskrétní maticí číselných hodnot. Pochopení této reprezentace je klíčové pro všechny následné operace.

1.1 Teoretický princip: Obraz jako matice a barevné prostory

V kontextu počítačového vidění je rastrový obraz (bitmapa) reprezentován jako mřížka pixelů. Každý pixel nese informaci o intenzitě světla v daném bodě. V nejběžnějším barevném modelu RGB (Red, Green, Blue) je každý pixel definován trojicí hodnot, obvykle v rozsahu 0 až 255 (pro 8bitovou hloubku), kde 0 představuje absenci barvy (černá) a 255 plnou intenzitu. Barevný obraz o rozměrech $H \times W$ (výška × šířka) je tedy matematicky reprezentován jako tenzor o rozměrech $H \times W \times 3$.4

Pro úlohu detekce tvarů, jako jsou mince, je však barva často redundantní a může dokonce vnášet do analýzy šum (například různé odlesky na zlaté a stříbrné minci mohou mást detektory hran). Proto je prvním krokem v našem řetězci převod do stupňů šedi (grayscale). V tomto jedkanálovém režimu ($H \times W \times 1$) reprezentuje hodnota pixelu jas (luminanci). Převod se obvykle neprovádí prostým průměrem RGB kanálů, ale váženým součtem, který zohledňuje citlivost lidského oka na různé vlnové délky světla (oko je nejcitlivější na zelenou):

```
Tento proces zjednodušuje data (redukce objemu dat na třetinu) a zdůrazňuje strukturální info

Dalším důležitým konceptem je barevný prostor **HSV (Hue, Saturation, Value)**. Zatímco RGB m

---

## 1.2 Redukce šumu a teorie filtrace

Digitální fotografie mincí nejsou nikdy dokonalé. Obsahují šum, který může pocházet z tepelné

Proto je nezbytné obraz **vyhladit (rozostřit)**.

### Gaussovské vyhlazování (Gaussian Blur)

Nejčastější metodou je konvoluce obrazu s Gaussovským jádrem. Jádro (kernel) je malá matice (
```

Parametr σ (směrodatná odchylka) určuje míru rozostření. Gaussovský filtr efektivně potlačuje vysokofrekvenční šum (náhodné pixely), ale má tendenci rozmazávat i hrany, což může být u mincí nežádoucí, pokud chceme přesně lokalizovat jejich okraj.8

Mediánový filtr (Median Blur)

Pro detekci mincí se často ukazuje jako vhodnější mediánový filtr. Na rozdíl od lineárního průměrování (Gaussian), mediánový filtr je nelineární operace, která nahrazuje hodnotu pixelu mediánem (prostřední hodnotou) intenzit v jeho okolí.

Tento filtr je excelentní v odstraňování impulsního šumu (tzv. "sůl a pepř" – bílé a černé tečky) a textury povrchu, přičemž má unikátní vlastnost: zachovává ostré hrany (edge-preserving). To je pro naši aplikaci klíčové, protože chceme odstranit ražbu uvnitř mince (aby nemátla detektor kruhů), ale zachovat ostrou hranici mezi mincí a pozadím.10

Porovnání filtrů pro předzpracování mincí

Typ filtru	Princip	Výhody	Nevýhody	Vhodnost pro mince
Průměrový (Box Blur)	Aritmetický průměr okolí	Velmi rychlý výpočet	Silně rozmazává hrany	Nízká
Gaussovský (Gaussian)	Vážený průměr dle Gaussovy křivky	Přirozené vyhlazení, odstranění Gaussova šumu	Rozmazává hrany (méně než Box)	Střední (dobrý pro Canny)
Mediánový (Median)	Medián hodnot okolí	Zachovává ostré hrany, odstraňuje texturu	Výpočetně náročnější	Vysoká (doporučeno)
Bilaterální (Bilateral)	Kombinace prostorové a intenzitní blízkosti	Nejlepší zachování hran	Velmi pomalý	Vysoká (ale často zbytečná)

1.3 Praktická implementace: Načtení a filtrace

Nyní převedeme teoretické poznatky do kódu. Využijeme knihovnu OpenCV (cv2), která je standardem pro zpracování obrazu v Pythonu.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_and_preprocess(image_path):
    """
    Načte obraz, převede ho do stupňů šedi a aplikuje mediánové vyhlazení.
    """
    # 1. Načtení obrazu ze souboru
    # Funkce imread načítá obraz v režimu BGR (Blue-Green-Red)
    img = cv2.imread(image_path)

    if img is None:
        raise FileNotFoundError(f"Soubor {image_path} nebyl nalezen.")

    # Vytvoření kopie pro pozdější vizualizaci, abychom neměnili originál
    output_image = img.copy()

    # 2. Převod do stupňů šedi (Grayscale)
```

```
    # Používáme konstantu cv2.COLOR_BGR2GRAY
    # Tím se zbavíme barevných informací, které nejsou pro detekci tvaru potřeba
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 3. Redukce šumu pomocí Mediánového filtru
    # Parametr '5' určuje velikost okna (kernelu) 5x5 pixelů.
    # Volba liché velikosti jádra je nutná pro existenci centrálního pixelu.
    # Větší jádro (např. 7, 9) více rozmaže, menší (3) zachová více detailů.
    # Pro mince je hodnota 5 obvykle dobrým kompromisem.
    gray_blurred = cv2.medianBlur(gray, 5)

    return img, gray_blurred, output_image

# Příklad použití (kód by byl součástí main bloku)
# original, processed, output = load_and_preprocess('coins.jpg')
```

V tomto kódu vidíme přímou aplikaci teorie: načtení dat, zahození nepotřebné dimenze (barvy) a aplikace nelineárního filtru pro potlačení textury mincí při zachování jejich obrysů.

Kapitola 2: Detekce hran a gradientní analýza

Po vyhlazení obrazu je naším cílem nalézt hranice objektů. Hrana v digitálním obraze je definována jako oblast s výraznou změnou jasu (diskontinuitou intenzitní funkce). Pokud se pohybujeme z tmavého pozadí na světlou minci, hodnota pixelů prudce vzroste.

2.1 Teoretický princip: Matematika gradientu

Pro detekci těchto změn používáme matematický nástroj zvaný gradient. Gradient obrazu $I(x,y)$ je vektor, který ukazuje směr největšího nárůstu jasu a jehož velikost odpovídá strmosti této změny.

```
Protože pracujeme s diskrétním obrazem, parciální derivace aproximujeme pomocí diferencí (roz

**Jádra Sobelova operátoru:**
```

Z těchto hodnot vypočítáme:

- Velikost gradientu (sílu hrany): $G = \sqrt{G_x^2 + G_y^2}$
- Směr gradientu (orientaci hrany): $\theta = \arctan\left(\frac{G_y}{G_x}\right)$

Směr gradientu je vždy kolmý na hranu. To je kriticky důležitý poznatek pro detekci kruhů, protože normály (směry gradientu) na obvodu kružnice se všechny protínají v jejím středu.13

2.2 Algoritmus Canny Edge Detector

Pro detekci hran mincí je průmyslovým standardem Cannyho detektor. Není to pouhý konvoluční filtr, ale vícestupňový algoritmus navržený tak, aby detekoval tenké, spojité hrany a minimalizoval falešné detekce. Skládá se z následujících kroků 15:

1. Gaussovské vyhlazení: (Již jsme provedli v předzpracování, ale Canny ho má interně také).
2. Výpočet gradientů: Pomocí Sobelova operátoru (viz výše).
3. Potlačení nemaximálních hodnot (Non-Maximum Suppression):

Tento krok "ztenčuje" hrany. Algoritmus prochází všechny pixely a zkoumá jejich gradient. Pokud hodnota gradientu pixelu není lokálním maximem ve směru gradientu, je hodnota nastavena na nulu. Výsledkem jsou hrany o šířce jednoho pixelu, což je nezbytné pro přesnou geometrickou lokalizaci mince.

4. Hysterezní prahování (Hysteresis Thresholding):

Toto je nejsofistikovanější část Cannyho detektoru. Místo jednoho prahu používá dva: T_{min} a T_{max}.

- Pixely s gradientem nad T_{max} jsou okamžitě označeny jako silné hrany.
- Pixely s gradientem pod T_{min} jsou zahozeny (nejsou hrany).
- Pixely mezi T_{min} a T_{max} jsou označeny jako slabé hrany. Tyto slabé hrany jsou ponechány ve výsledném obraze pouze tehdy, pokud prostorově navazují na silnou hranu.

Tento mechanismus hystereze je pro mince zásadní. Odlesk na hraně mince může být v některých místech silný a jinde slabý (kvůli stínu). Jednoduché prahování by kruh přerušilo. Hystereze umožní detekovat celý obvod mince, pokud je alespoň část hrany výrazná.16

Kapitola 3: Geometrická detekce kružnic (Houghova transformace)

Máme-li obraz hran, dalším úkolem je nalézt v těchto hranách geometrické primitivum – kružnici. K tomu slouží Houghova transformace (Hough Transform, HT), která převádí problém detekce z obrazového prostoru do prostoru parametrů.

3.1 Teoretický princip: Od obrazu k parametrům

Analytická rovnice kružnice je:

```
def detect_circles_hough(gray_image):
```

```
    """
    Detekuje kružnice v šedotónovém obraze pomocí Hough Gradient Method.
    """
    circles = cv2.HoughCircles(
        image=gray_image,
        method=cv2.HOUGH_GRADIENT,
        dp=1.2,           # Rozlišení akumulátoru
        minDist=50,       # Minimální vzdálenost mezi středy kružnic
        param1=50,        # Práh pro Cannyho detektor (horní mez)
        param2=30,        # Práh pro akumulátor středů (počet hlasů)
        minRadius=15,     # Minimální poloměr mince (v pixelech)
        maxRadius=120     # Maximální poloměr mince (v pixelech)
    )

    if circles is not None:
        # Převedeme souřadnice na celá čísla (pro vykreslení)
        circles = np.uint16(np.around(circles))
        return circles[0, :] # Funkce vrací 3D pole, nás zajímá jen list kruhů
    return

def segment_watershed(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Odstranění šumu
    kernel = np.ones((3,3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

    # Jisté pozadí (Sure Background) - dilatace objektu
    sure_bg = cv2.dilate(opening, kernel, iterations=3)

    # Jisté popředí (Sure Foreground) - Distance Transform
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

    # Neznámá oblast (Unknown region)
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)

    # Vytvoření markerů
    ret, markers = cv2.connectedComponents(sure_fg)
    markers = markers + 1 # Posun, aby pozadí bylo 1, ne 0
    markers[unknown == 255] = 0 # Neznámá oblast je 0

    # Aplikace Watershed
    markers = cv2.watershed(img, markers)
    img[markers == -1] =  # Vykreslení hranic červeně
    return img
```

kde (a, b) jsou souřadnice středu a r je poloměr. Máme tedy tři neznámé parametry: $a, b,

V klasické Houghově transformaci bychom vytvořili 3D akumulátor (pole), kde osy odpovídají tě

Proces "hlasování" (voting) funguje následovně:

* Pro každý hranový bod (x_i, y_i) v obraze si představíme všechny možné kružnice, které by
* V akumulátoru inkrementujeme hodnotu všech buněk (a, b, r), které vyhovují rovnici kružni
* Pokud body v obraze skutečně tvoří kružnici, jejich kuželové plochy v prostoru parametrů se

Analogicky si to lze představit jako sčítání kruhů. Pokud kolem každého bodu hrany narýsujeme

---

3.2 Hough Gradient Method: Optimalizace pro praxi

Klasická 3D Houghova transformace je výpočetně a paměťově extrémně náročná (složitost roste s

Algoritmus probíhá ve dvou fázích:

1. **Fáze 1: Detekce středů (2D akumulátor):**
   Víme, že gradient na hraně kružnice směřuje do jejího středu. Pro každý hranový bod tedy s

2. **Fáze 2: Odhad poloměru (1D histogram):**
   Jakmile máme kandidáty na středy, algoritmus vypočítá vzdálenosti od tohoto středu ke všem

Tato metoda je mnohem rychlejší, ale nese s sebou specifické vlastnosti: je citlivější na šum

---

3.3 Praktická implementace: Funkce HoughCircles

V OpenCV je tato sofistikovaná logika zabalena do funkce `cv2.HoughCircles`. Její parametry j

Vysvětlení klíčových parametrů 19:

* **dp (Inverse Ratio of Resolution):** Toto číslo určuje, jak jemná je mřížka akumulátoru op

  * `dp=1`: Akumulátor má stejné rozlišení jako obraz. Nejpřesnější, ale nejpomalejší.
  * `dp=2`: Akumulátor má poloviční šířku a výšku. Rychlejší, ale může slít blízké středy.
    Doporučení: Hodnoty mezi 1.0 a 1.5 jsou pro mince ideální.

* **minDist (Minimum Distance):** Minimální eukleidovská vzdálenost mezi středy detekovaných

  * Příliš malá: může detekovat více kružnic na jedné minci (např. vnitřní a vnější okraj raž
  * Příliš velká: může minout mince, které leží blízko u sebe.

* **param1 (Canny Threshold):** Horní práh (T_{max}) pro Cannyho detektor hran, který Hough

  * Vysoká hodnota (>100): detekuje jen velmi ostré hrany.
  * Nízká hodnota (<50): detekuje i slabé hrany, ale zvyšuje riziko falešných kruhů z šumu.

* **param2 (Accumulator Threshold):** Kritický parametr. Určuje, kolik "hlasů" (průsečíků gra

  * Příliš nízko: detekuje falešné kruhy v textuře koberce nebo stolu.
  * Příliš vysoko: nenajde mince s méně výrazným okrajem.
    Strategie ladění: Začněte s vysokou hodnotou (např. 100) a snižujte ji, dokud nejsou

---

Kapitola 4: Pokročilá segmentace překrývajících se mincí (Watershed)

Houghova transformace je robustní pro izolované mince. Co se ale stane, když se mince dotýkaj

---

4.1 Teoretický princip: Obraz jako krajina

Watershed algoritmus chápe obraz (v odstínech šedi) jako topografickou mapu, kde jas pixelu o

* Tmavé oblasti (hrany a mezery) jsou "pohoří".

* Světlé oblasti (středy mincí) jsou "údolí" nebo "pánve".

Představme si, že do každého lokálního minima (střed mince) začneme napouštět vodu. Hladina s

Aby Watershed fungoval správně a "nepřehradil" obraz kvůli šumu (tzv. oversegmentation), musí

4.2 Distance Transform a tvorba markerů

Jak získat jisté markery pro mince? Použijeme **Distance Transform (Transformaci vzdálenosti)

1. Provedeme binární prahování (Otsu method), abychom získali hrubé obrysy mincí.
2. Aplikujeme Distance Transform: Každému pixelu popředí přiřadíme hodnotu odpovídající jeho
3. Středy mincí budou mít nejvyšší hodnotu (jsou nejdále od okraje).
4. Prahováním této mapy vzdáleností získáme "jisté popředí" (Sure Foreground) – středy mincí,
5. Oblast mezi jistým popředím a jistým pozadím je "neznámá oblast" (Unknown Region) – právě

4.3 Implementační detail

Tento přístup je pro studenty fascinující ukázkou, jak lze kombinací morfologických operací a

---

Kapitola 5: Úvod do Hlubokého učení (Deep Learning) pro klasifikaci

Zatímco Houghova transformace a Watershed nám řekly, kde mince jsou (lokalizace), nedokážou n

---

5.1 Od perceptronu ke konvoluci

Klasické neuronové sítě (MLP - Multi-Layer Perceptron) jsou pro zpracování obrazu nevhodné. P

CNN řeší tento problém inspirací z biologie (zraková kůra koček, experimenty Hubela a Wiesela

---

5.2 Stavební bloky CNN

Porozumění CNN vyžaduje pochopení tří hlavních typů vrstev, kterými obraz prochází 4:

1. Konvoluční vrstva (Convolutional Layer) – Extraktor příznaků

Toto je srdce sítě. Místo pevných vah používá síť filtry (kernely) – malé matice (např. $3 \t

* V prvních vrstvách se filtry naučí detekovat jednoduché struktury: svislé hrany, vodorovné
* V hlubších vrstvách sítě kombinují tyto jednoduché příznaky do složitějších vzorů: kruhové

Analogie: Představte si filtry jako sadu specializovaných brýlí. Jedny vidí jen svislé čáry,

2. Aktivační funkce (ReLU) – Zavedení nelinearity

Výstup z konvoluce je lineární operace. Aby síť mohla modelovat složité jevy, potřebujeme nel

Tato funkce jednoduše nahradí všechny záporné hodnoty nulou. Funguje jako "práh" – pokud je příznak (např. hrana) detekován slabě nebo vůbec (záporná nebo nízká hodnota), neuron se neaktivuje. To umožňuje síti ignorovat irelevantní informace a soustředit se na silné signály.30

3. Pooling vrstva (Max Pooling) – Redukce dimenze

Tato vrstva slouží ke zmenšení obrazu (downsampling). Obvykle se používá okno 2×2 s krokem 2, ze kterého se vybere maximální hodnota.

- Význam: Snižuje výpočetní náročnost (méně dat pro další vrstvy).
- Invariance: Činí síť robustní vůči malým posunům a deformacím. Pokud se mince v obraze posune o pixel doleva, maximální hodnota v okně 2×2 zůstane pravděpodobně stejná. Síť tak "pozná" minci, i když není dokonale vycentrovaná.27

4. Plně propojená vrstva (Fully Connected / Dense Layer) – Klasifikátor

Po průchodu mnoha konvolučními a pooling vrstvami získáme sadu abstraktních příznaků (např. "přítomnost kruhu", "přítomnost číslice 5", "stříbrná barva"). Tato data jsou zploštěna do vektoru a předána klasické neuronové síti, která na základě těchto příznaků rozhodne o finální třídě (např. s pravděpodobností 98 % jde o 5 Kč).

Kapitola 6: Architektury CNN a Trénink modelu

V praxi málokdy navrhujeme vlastní architekturu CNN od nuly. Využíváme ověřené architektury, které dosahují špičkových výsledků (State-of-the-Art). Pro klasifikaci mincí jsou relevantní zejména ResNet a MobileNet.31

6.1 Porovnání architektur pro mince

Architektura	Charakteristika	Výhody	Nevýhody	Vhodné použití
ResNet-50	Hluboká síť s reziduálními spoji (skip connections). Řeší problém mizejícího gradientu tím, že signál může "přeskočit" vrstvy.	Vysoká přesnost, schopnost učit se velmi jemné detaily (ražba).	Výpočetně náročnější, pomalejší inference, větší model na disku.	Serverové aplikace, vysoká přesnost vyžadována.

MobileNetV2	Optimalizovaná pro mobilní zařízení. Využívá "Depthwise Separable Convolutions", které rozdělují konvoluci na prostorovou a hloubkovou.	Extrémně rychlá, malá paměťová náročnost, běží i na CPU/Raspberry Pi.	Mírně nižší přesnost než ResNet u velmi složitých úloh.	Mobilní aplikace, real-time detekce na slabém HW.
YOLO (v8/v11)	"You Only Look Once". Jednostupňový detektor. Nedělá výřez, ale rovnou detekuje bounding boxy i třídy v celém obraze.	Nejrychlejší (real-time video), řeší detekci i klasifikaci naráz.	Vyžaduje trénovací data anotovaná bounding boxy (složitější příprava dat).	Detekce z videa, robotika.

Pro naši výukovou aplikaci, která kombinuje Houghovu transformaci (pro detekci) a CNN (pro klasifikaci výřezu), je ideální volbou MobileNetV2 nebo ResNet-18/50 díky jejich dostupnosti v knihovnách torchvision a tensorflow.keras.32

6.2 Data: Palivo pro neuronovou síť

Kvalita modelu závisí na datech. Pro mince existují datasety na Kaggle (např. 211 tříd z 32 měn) 34, ale často musíme vytvořit vlastní dataset. Zde narážíme na problém přeurečení (overfitting) a potřebu augmentace dat.

Datová augmentace (Data Augmentation)

Protože nemůžeme nafotit minci ve všech možných polohách a za všech světelných podmínek, uměle generujeme nové trénovací vzorky transformací těch existujících.

Pro mince jsou klíčové tyto transformace 36:

- Rotace (Random Rotation): Mince je kruhově symetrická, ale ražba má orientaci. Model musí poznat "5" i když je hlavou dolů. Rotujeme o náhodný úhel v rozsahu $0-360^\circ$.
- Jas a Kontrast (Color Jitter): Simuluje různé osvětlení a odlesky kovu.
- Posun a Škálování: Simuluje nepřesnost detekce výřezu (Houghova transformace nemusí být pixel-perfect).
- Varování: Flip (Překlopení/Zrcadlení) je u mincí problematické. Pokud zrcadlíte obraz, číslice a písmena budou zrcadlově obrácená, což v realitě nenastává. Síť by se mohla naučit nesmyslné vzory. Používejte opatrně.

6.3 Interpretace tréninkových křivek (Loss Curves)

Během tréninku sledujeme dvě metriky: Training Loss (chyba na trénovacích datech) a Validation Loss (chyba na datech, která síť nikdy neviděla). Pochopení těchto křivek je pro studenty zásadní diagnostickou dovedností 39:

- Underfitting (Podučení): Obě křivky (Train i Val Loss) jsou vysoké a neklesají. Model je příliš jednoduchý nebo je špatně nastavený learning rate.
- Overfitting (Přeurečení): Training Loss klesá k nule (model se "bifluje" data nazpaměť), ale Validation Loss začne v určitém bodě stoupat. Model se nenaučil obecné rysy mince, ale šum v tréninkové sadě.
- Řešení: Více dat, silnější augmentace, Dropout (náhodné vypínání neuronů), Early Stopping (zastavení tréninku v bodě zvratu).
- Good Fit (Ideální stav): Obě křivky klesají a stabilizují se s malým rozdílem mezi nimi.

Kapitola 7: Implementační příručka (Code Walkthrough)

V této části spojíme teorii do funkčního celku. Použijeme hybridní přístup: OpenCV pro nalezení mince a její vyříznutí, a PyTorch s modelem ResNet pro klasifikaci.

7.1 Krok 1: Lokalizace a extrakce (OpenCV)

Nejprve definujeme funkci, která najde mince a vrátí jejich výřezy (Region of Interest - ROI).

```
import cv2
import numpy as np

def extract_coins(image_path, debug=False):
    """
    Nalezne mince v obraze a vrátí jejich výřezy a souřadnice.
    """
    # 1. Načtení a předzpracování
    img = cv2.imread(image_path)
    if img is None: return,
    original = img.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Použití mediánového filtru pro odstranění textury
    gray_blur = cv2.medianBlur(gray, 5)

    # 2. Houghova transformace
    # Parametry je nutné ladit podle rozlišení kamery a velikosti mincí
    circles = cv2.HoughCircles(
        gray_blur,
        cv2.HoughModes.HOUGH_GRADIENT,
        dp=1.2,
        minDist=60,         # Mince nesmí být blíž než 60px
        param1=50,          # Canny high threshold
        param2=35,          # Accumulator threshold (citlivost)
        minRadius=20,
        maxRadius=150
    )
```

```
    rois =
    coords =

    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            x, y, r = i, i, i

            # 3. Bezpečný výřez (kontrola hranic obrazu)
            # Přidáme malý padding, abychom měli celou minci
            pad = 0
            y1 = max(0, y - r - pad)
            y2 = min(img.shape, y + r + pad)
            x1 = max(0, x - r - pad)
            x2 = min(img.shape, x + r + pad)

            roi = original[y1:y2, x1:x2]

            # Kontrola, zda má výřez smysluplnou velikost
            if roi.shape > 10 and roi.shape > 10:
                rois.append(roi)
                coords.append((x, y, r))

    return rois, coords
```

7.2 Krok 2: Příprava modelu CNN (PyTorch)

Zde použijeme techniku Transfer Learning. Vezmeme model ResNet18 předtrénovaný na obrovském datasetu ImageNet (umí rozeznávat hrany, textury) a "přeučíme" pouze jeho poslední vrstvu na naše třídy mincí.32

```
import torch
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image

def get_coin_model(num_classes, model_path=None):
    # Načtení předtrénovaného ResNet18
    model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

    # Změna poslední plně propojené vrstvy
    # ResNet má na vstupu do poslední vrstvy 512 příznaků
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, num_classes)

    # Pokud máme uložené váhy z vlastního tréninku, načteme je
    if model_path:
        model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))

    model.eval() # Přepnutí do evaluačního režimu (vypne Dropout a Batch Norm training)
    return model

def classify_roi(roi_cv2, model, class_names, device):
    # Definice transformací - musí odpovídat tomu, jak byl model trénován
    preprocess = transforms.Compose(, std=[0.229, 0.224, 0.225]),
    ])
```

```
    input_tensor = preprocess(roi_cv2)
    # Přidání dimenze dávky (Batch dimension): ->
    input_batch = input_tensor.unsqueeze(0).to(device)

    with torch.no_grad(): # Vypnutí gradientů pro inferenci (rychlost)
        output = model(input_batch)

    # Získání pravděpodobností pomocí Softmax
    probabilities = torch.nn.functional.softmax(output, dim=0)

    # Získání třídy s nejvyšší pravděpodobností
    prob, predicted_class_idx = torch.topk(probabilities, 1)

    label = class_names[predicted_class_idx.item()]
    confidence = prob.item()

    return label, confidence
```

7.3 Krok 3: Spojení pipeline a vizualizace

Nyní vytvoříme hlavní smyčku, která vezme obraz, najde mince, klasifikuje je a vykreslí výsledek.

```
def main_pipeline(image_path, model_path, class_names):
    # Nastavení zařízení (GPU pokud je dostupné, jinak CPU)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # 1. Načtení modelu
    model = get_coin_model(len(class_names), model_path)
    model.to(device)

    # 2. Detekce mincí (OpenCV)
    rois, coords = extract_coins(image_path)

    # Načtení obrazu pro kreslení výsledků
    result_img = cv2.imread(image_path)

    total_value = 0.0

    print(f"Nalezeno {len(rois)} kandidátů na mince.")

    # 3. Klasifikace každé mince
    for i, roi in enumerate(rois):
        label, conf = classify_roi(roi, model, class_names, device)
        x, y, r = coords[i]

        # Filtrování nízké jistoty (např. falešné kruhy)
        if conf > 0.6:
            # Vykreslení kruhu a textu
            cv2.circle(result_img, (x, y), r, (0, 255, 0), 4)
            cv2.circle(result_img, (x, y), 2, (0, 0, 255), 3)

            text = f"{label} ({conf:.2f})"
            cv2.putText(result_img, text, (x - 40, y - r - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
```

```
            # Sčítání hodnoty (vyžaduje mapování label -> hodnota)
            # total_value += coin_values[label]
        else:
            # Vykreslení zahozené detekce červeně
            cv2.circle(result_img, (x, y), r, (0, 0, 255), 2)

    # Zobrazení výsledku
    plt.imshow(cv2.cvtColor(result_img, cv2.COLOR_BGR2RGB))
    plt.show()

# Příklad definice tříd (musí odpovídat tréninku)
# classes = ['1kc', '2kc', '5kc', '10kc', '20kc', '50kc']
```

Kapitola 8: Výzvy, limity a nasazení aplikace

Ačkoli výše uvedený kód funguje v kontrolovaných podmínkách, reálný svět přináší komplikace. Studenti by měli být seznámeni s limity technologie.

8.1 Problémy Houghovy transformace

- Falešné pozitivy: Kruhové logo na předmětu, kulatá skvrna na stole nebo hlavička šroubu mohou být detekovány jako mince. Řešením je trénovat CNN s třídou "background/not_coin", aby tyto falešné detekce v druhém kroku vyřadila.
- Překryv mincí: Jak bylo zmíněno v kapitole 4, Houghova transformace selhává u překrývajících se mincí. Zde je nutné integrovat Watershed algoritmus do pipeline před extrakcí ROI.

8.2 Vliv osvětlení a rotace

CNN jsou citlivé na změnu distribuce dat (domain shift). Pokud model natrénujete na fotkách s bleskem a testujete na fotkách za šera, přesnost klesne. Řešením je masivní augmentace jasu/kontrastu při tréninku a normalizace histogramu (Histogram Equalization) při předzpracování.37

8.3 Nasazení aplikace: Webové rozhraní (Streamlit)

Pro prezentaci studentského projektu je ideální vytvořit jednoduché webové rozhraní. Knihovna Streamlit umožňuje převést náš Python skript na webovou aplikaci bez znalosti HTML/CSS.45

Struktura aplikace `app.py`:

- Uživatel nahraje obrázek (`st.file_uploader`).

- Aplikace zavolá naši funkci `main_pipeline`.
- Výsledek se zobrazí pomocí `st.image`.
- Lze přidat posuvníky (sliders) pro dynamické ladění parametrů Houghovy transformace (`param1`, `param2`), což je vynikající výuková pomůcka pro pochopení jejich vlivu v reálném čase.

Závěr

Tato příručka provedla čtenáře kompletní cestou automatizované analýzy mincí. Ukázali jsme, že zdánlivě jednoduchá úloha "spočítat drobné" v sobě skrývá hluboké matematické koncepty – od gradientů a konvolucí, přes statistickou analýzu v Houghově prostoru, až po nelineární transformace v hlubokých neuronových sítích.

Kombinace deterministického počítačového vidění (OpenCV) pro lokalizaci a pravděpodobnostního hlubokého učení (PyTorch) pro klasifikaci představuje robustní a moderní přístup, který je dnes standardem v průmyslových aplikacích. Doufáme, že tento materiál poslouží studentům jako pevný základ pro další experimenty v fascinujícím oboru počítačového vidění.