Kapitola 3

Konvoluční neuronové sítě

V této kapitole se zaměříme na konvoluční neuronové sítě (CNN), které patří mezi nejpoužívanější modely pro zpracování obrázků. Jejich hlavní výhodou je, že si dokážou z obrazových dat postupně vytvářet představu o tom, co na obrázku je – nejprve zachytí jednoduché prvky, jako jsou hrany a barvy, a následně z nich skládají složitější tvary a objekty. CNN se díky tomu uplatňují v celé řadě úloh spojených s vizuálním vnímáním, například při rozpoznávání obličejů, třídění fotografií nebo při řízení autonomních vozidel. Tento typ sítě představuje důležitý krok k porozumění tomu, jak počítače „vidí" a jak mohou napodobovat lidské vnímání obrazu.

Cíle kapitoly *(po přečtení této kapitoly budete schopni)*

C1.1. Vysvětlit, proč klasické neuronoví sítě nemusí být vhodné pro zpracování obrázků a proč vznikly konvoluční neuronové sítě (CNN).

C1.2. Popsat základní princip fungování CNN a vysvětlit, jak CNN prohlíží obrázek po menších částech pomocí filtrů.

C1.3. Popsat, jak počítač reprezentuje obrázek jako mřížku čísel a proč je důležité zachovat prostorovou strukturu pixelů.

C1.4. Vyjmenovat hlavní stavební vrstvy CNN a stručně vysvětlit jejich funkci.

C1.5. Porozumět celému procesu zpracování obrázku v CNN – od vstupního obrázku až po konečné rozhodnutí.

C1.6. Uvést základní oblasti, kde se CNN používají, a popsat možnosti, jak s CNN začít pracovat v praxi.

Klíčové myšlenky *(po přečtení této kapitoly byste si měli pamatovat)*

KM1.1. Obrázek je pro počítač jen mřížka čísel (pixelů), nikoli objekt jako pro člověka. Pro pochopení obrázků je nutné zachovat vztahy mezi sousedními pixely.

KM1.2. Konvoluční neuronové sítě dokážou pracovat s obrázky efektivněji než klasické neuronové sítě, protože prohlížejí obrázek po menších částech pomocí filtrů, a tím nacházejí rysy a vzory.

KM1.3. Konvoluční, „pooling" a další vrstvy postupně extrahují z obrázku rysy: od hran přes tvary až po celé objekty.

KM1.4. CNN se učí podobně jako jiné neuronové sítě – porovnává svůj vstup se správnou odpovědí, vypočítá chybu a mění své parametry pomocí zpětné propagace.

KM1.5. CNN mají široké využití v praxi a dnes existují snadné způsoby, jak si je vyzkoušet i bez složité instalace programů.

Nové příkazy *(po přečtení kapitoly se seznámím s těmito novými příkazy)*

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

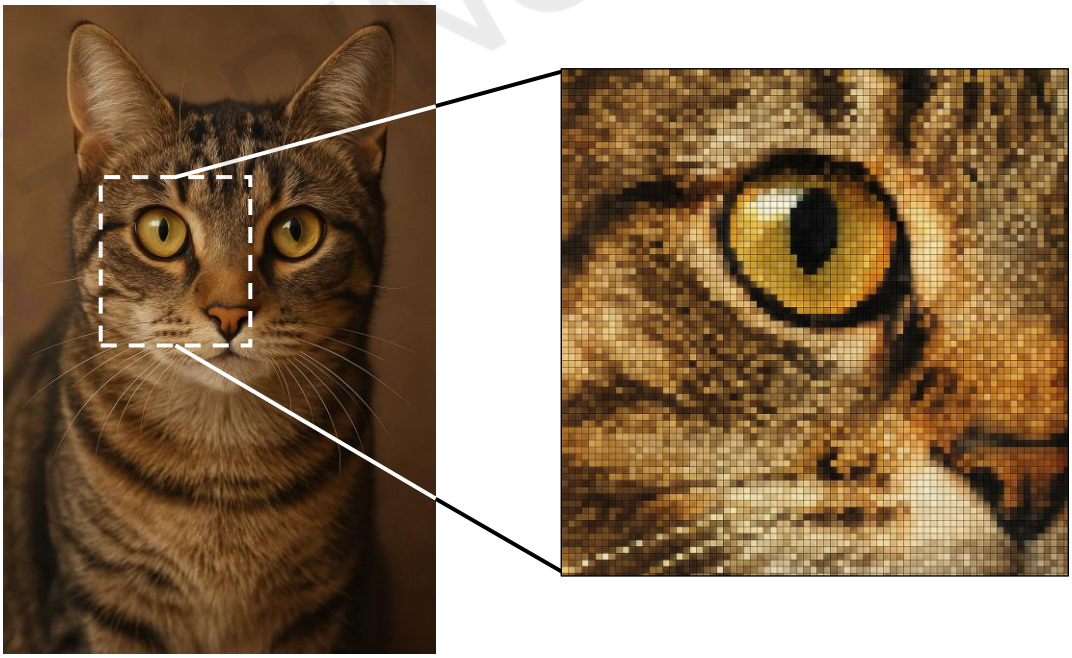
1.1 Proč existují konvoluční neuronové sítě?

Nabízí se otázka: proč vůbec existují neuronové sítě zvané konvoluční a k čemu se používají?

Dříve, než vznikly konvoluční neuronové sítě (také označované jako CNN), se výzkumníci pokoušeli zpracovávat obrázky pomocí běžných neuronových sítí, podobných těm, o kterých jsme si něco málo pověděli u neuronu. U malých a jednoduchých příkladů to mohlo fungovat, ale jakmile přišla řada na skutečné fotografie, brzy narazili na problém.

Počítač totiž nevidí obrázek tak jako člověk, pro něj je obrázek jen velká tabulka čísel. Každé číslo představuje jeden „světelný bod“, tedy pixel – ten je dán svou pozicí a barvou, viz Obrázek 1, který obsahuje detailní pohled na složení obrázku kočky z jednotlivých barevných pixelů. Například černobílý obrázek o velikosti 100 x 100 pixelů obsahuje 100 x 100, tedy 10 000 takových čísel. U barevného obrázku je to ještě více – počítač si totiž musí pamatovat intenzitu jednotlivých barev (nejčastěji se používají tři hodnoty pro každý pixel, které kombinují, jak moc je v něm červená, zelená a modrá barva). Místo 10 000 hodnot se tak najednou musí pracovat s 30 000 hodnotami.

Pokud bychom takový obrázek poslali do klasické neuronové sítě, znamenalo by to, že síť musí pracovat s tisíci nebo i miliony čísel najednou. To je jednak velmi pomalé, ale hlavně je pro tuto síť nesmírně těžké se v takovém množství dat „zorientovat“. A navíc je tu ještě jeden zásadní problém: běžná neuronová síť s těmito čísly zachází, jako by to byl pouze dlouhý seznam hodnot, aniž by chápala, že některá čísla patří k sobě, protože představují sousední pixely nacházející se v obrázku. Pro lepší představu se podívejte na obrázek kočky, který se nachází níže. Nám lidem je jasné, že pixely tvořící oko spolu souvisejí, protože jsou vedle sebe a společně právě vytvářejí určitý tvar oka. Klasická neuronová síť to ale neví – pro ni je každý pixel izolovaná číselná informace a nedokáže přirozeně využít skutečnost, že okolní pixely spolu vytvářejí důležité obrazce.



Obrázek 1: Ilustrační ukázka pixelů na detailu obrázku kočky.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Výzkumníci proto potřebovali vyřešit dvě zásadní věci:

1. Jak vytvořit takovou neuronovou síť, která by chápala prostorovou strukturu obrázku – tedy že pixely postupně tvoří tvary, hrany a objekty?
2. Jak zmenšit množství parametrů – tedy čísel, se kterými síť musí pracovat – tak, aby nebyla příliš velká a šla prakticky trénovat?

Konvoluční neuronové sítě vznikly právě jako odpověď na tyto problémy. Byly navrženy tak, aby se dokázaly dívat na obrázek podobně jako člověk – postupně, pomenších částech, a aby v nich dokázaly hledat důležité vzory (oči, uši, ocas apod.). Díky tomu se CNN dokáže naučit rozpoznat hrany, tvary, a nakonec i celé objekty, a to mnohem efektivněji než klasická neuronová síť.

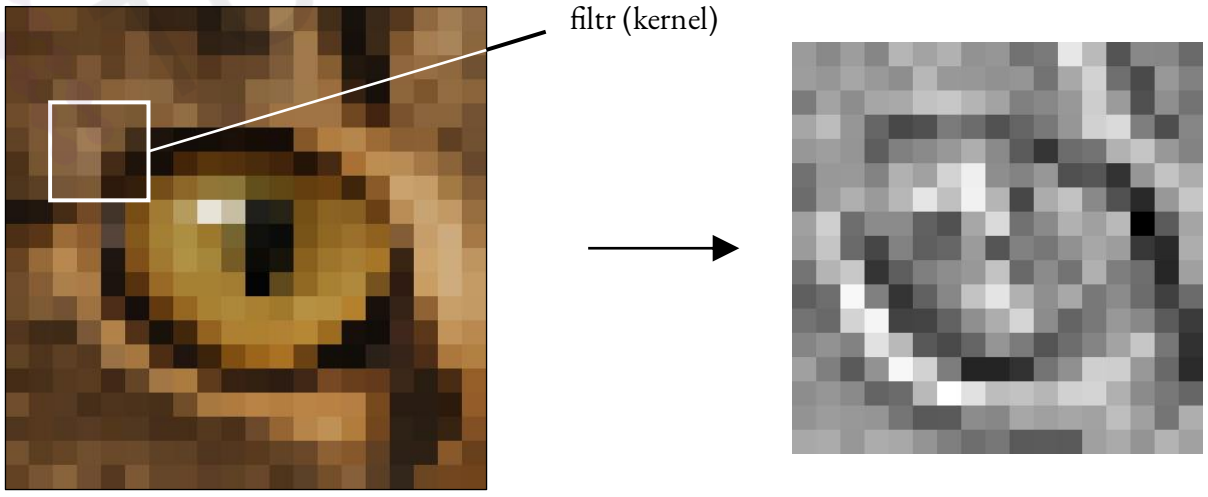
Dnes jsou konvoluční neuronové sítě základem většiny systémů pro zpracování obrazů a videa. Umožnily pokrok v oblastech, jako je rozpoznávání obličejů v mobilních telefonech, autonomní vozidla nebo například analýza lékařských snímků.

1.2 Co je to konvoluční neuronová síť?

V předchozí části jsme si vysvětlili, proč je pro práci s obrázky potřeba používat speciální typ neuronové sítě. Nyní se podíváme na to, co je to vlastně konvoluční neuronová síť (CNN) a jaký je její základní princip.

Konvoluční neuronová síť je druh umělé neuronové sítě, který je navržen tak, aby dokázal zpracovávat data, ve kterých hraje roli prostorové uspořádání jednotlivých prvků – například právě obrázky. Zatímco klasická neuronová síť přijímá všechny vstupy najednou a zpracovává je jako jeden dlouhý seznam čísel, CNN pracuje s obrázkem postupně, po jeho menších částech. Tím dokáže zachytit důležité vztahy mezi pixely, které jsou blízko sebe, a učí se je rozpoznávat jako určité vzory.

Můžeme si to představit jako postupné **prohlížení obrázku přes malé okénko**, které se po obrázku posouvá a zjišťuje, zda se v jednotlivých částech nenachází nějaký typický tvar nebo prvek (například hrana, kousek ucha, oko nebo jiný detail, viz obrázek níže). Tento malý „průzkumník“ se nazývá filtr (nebo také kernel).

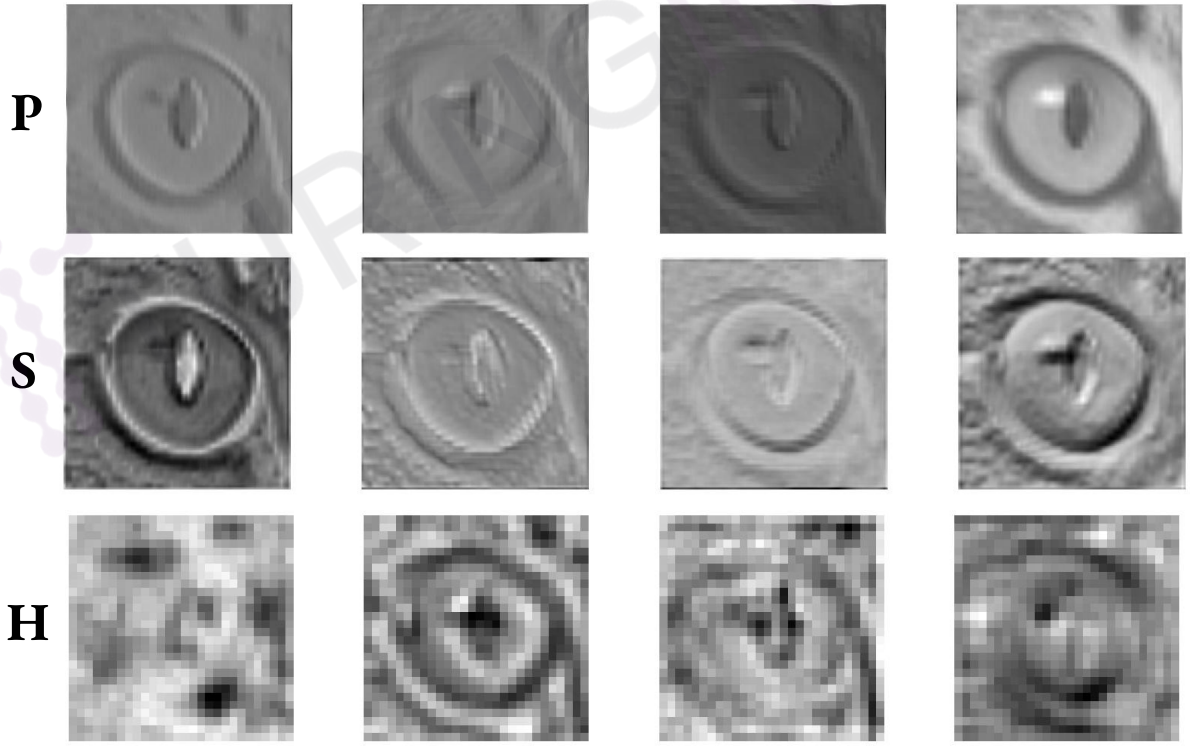


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Velkou výhodou tohoto přístupu je, že se stejný filtr aplikuje na celý obrázek, a to znamená dvě klíčové věci:

1. Síť může najít ten samý prvek (například oko) kdekoliv v obrázku, nejen na jednom místě.
2. Není potřeba mít tolik parametrů jako v klasické neuronové síti, protože se ten stejný filtr používá opakovaně.

CNN tedy nepracuje s každým pixelem zvlášť, ale učí se hledat a kombinovat vzory, které jsou pro obrázek charakteristické. A čím více vrstev síť má, tím složitější vzory je schopna rozeznat. V prvních vrstvách (viz obrázek níže P) obvykle rozpoznává například hrany, v dalších (viz obrázek níže S) už složitější tvary, a ve vyšších vrstvách (viz obrázek níže H) dokonce celé objekty (například oko, ucho, nebo rovnou celou kočku).



Konvoluční neuronová síť tedy kombinuje dvě důležité myšlenky:

1. dívá se na obrázek po částech a hledá v něm určité vzory a
2. tyto vzory dále zpracovává, skládá a využívá k rozpoznání toho, co se na obrázku nachází.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

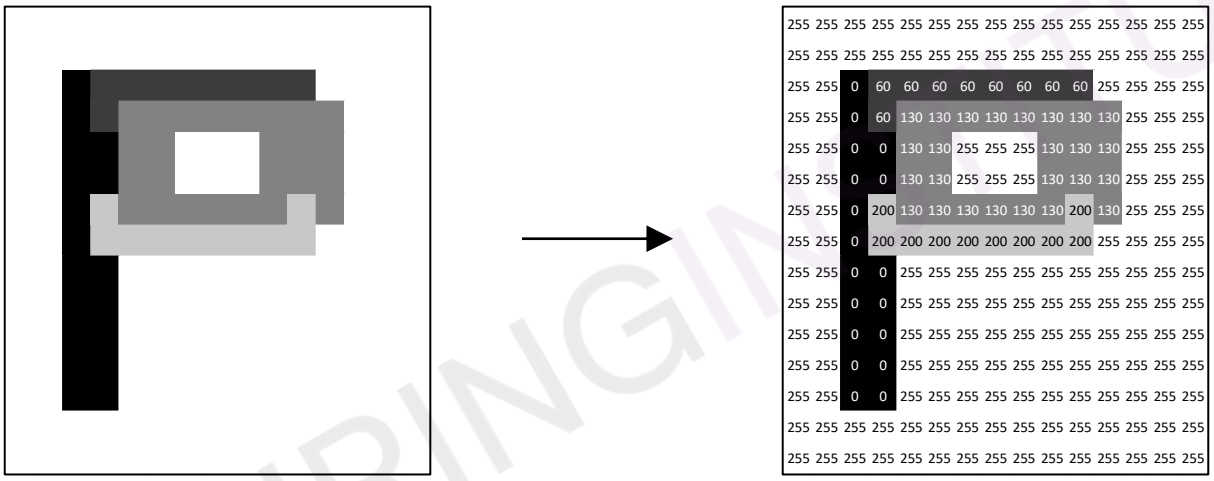
Výsledkem je, že CNN s obrázky dokáže pracovat podobně jako lidské oko a mozek – nejdříve zachytí drobné detaily a postupně si z nich skládá celkový obraz.

1.3 Jak počítač vnímá obrázek?

Abychom dokázali pochopit, jak konvoluční neuronová síť pracuje, je potřeba nejdříve porozumět tomu, jak počítač vlastně „vidí“ obrázek. My lidé vnímáme obrázek jak ocelek – vidíme objekty, barvy, tvary a rozpoznáváme, co se na obrázku nachází. Počítač však žádný obraz nevidí v pravém slova smyslu. Pro něj je obrázek pouze soubor čísel, která popisují, jak světlý nebo barevný je každý jednotlivý bod obrázku, který se nazývá pixel.

Obrázek jako mřížka čísel

Obrázek si můžeme představit jako obrovskou tabulku (mřížku) buněk, kde v každé buňce se nachází číslo. Pokud se jedná o černobílý obrázek, obsahuje každý pixel pouze jednu hodnotu, která obvykle udává jas (například číslo od 0 do 255, kde 0 reprezentuje černou barvu a 255 barvu bílou, viz obrázek níže).

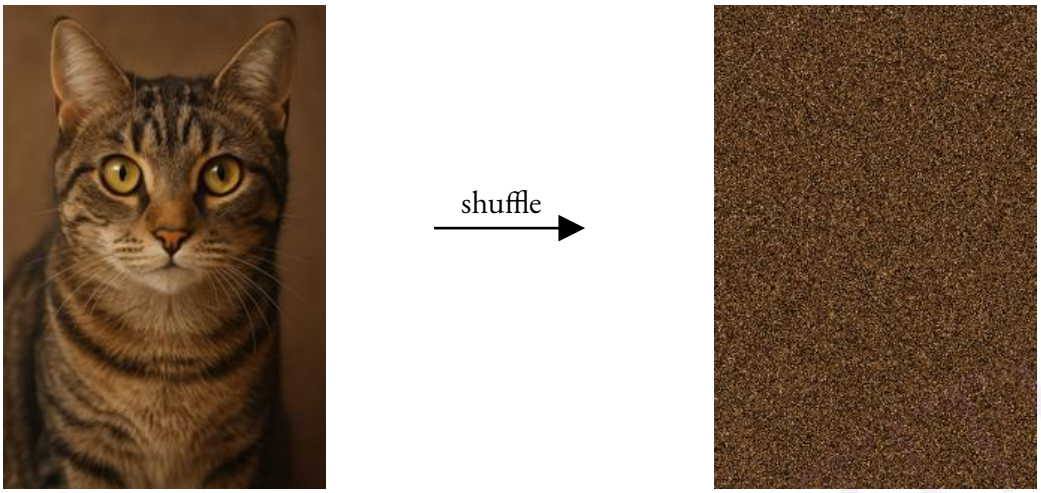


U barevného obrázku je situace podobná, jen místo jedné hodnoty má každý pixel hodnoty tři. Tyto tří hodnoty představují intenzitu tří základních barev – červené (R), zelené (G) a modré (B). Jejich kombinací lze totiž vytvořit všechny barvy, které na běžném obrázku vidíme. Zatímco tedy malý černobílý obrázek o velikosti 10 x 10 pixelů obsahuje 100 čísel, barevný obrázek stejné velikosti obsahuje 300 čísel.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Prostorová struktura obrázku

Důležitým faktem je to, že pixely nejsou náhodně poskládané hodnoty. Jejich uspořádání má smysl – sousední pixely spolu vytvářejí určitou část obrázku – například roh, tvar nebo přechod barvy. Pokud byste pixely náhodně promíchali (viz obrázek níže vpravo), vznikla by neuspořádaná změť barev, ve které by nebylo možné rozeznat žádný objekt. Člověk by v takovém obraze nepoznal vůbec nic – a stejně tak asi ani počítač.



A právě proto je důležité, aby neuronová síť uměla pracovat s obrázkem v jeho prostorové podobě, a nejen jako s dlouhým seznamem čísel. Právě na touto schopností disponují takzvané konvoluční vrstvy, o kterých si podrobněji povíme v dalších kapitolách.

1.4 Hlavní stavební části CNN

V předchozích částech jsme si vysvětlili, proč konvoluční sítě vznikly a v čem se liší od klasických neuronových sítí. Nyní se podíváme na to, z jakých částí se CNN nejčastěji skládá. Tyto části se nazývají vrstvy a každá z nich má v síti svou specifickou úlohu. Konvoluční neuronová síť je tvořena několika typy vrstev, které spolupracují tak, aby z obrázku dokázaly získat užitečné informace a nakonec určit, co se na obrázku nachází. Níže si jednotlivé vrstvy postupně představíme.

Konvoluční vrstvy

Konvoluční vrstva je základním stavebním kamenem CNN. Obsahuje filtry (kernely) – malé mřížky čísel, které procházejí obrázek po menších částech. Filtr se vždy položí na malou oblast obrázku (například 3x3 oblast pixelů) a provede operaci, ve které hodnoty pixelů s hodnotami ve filtru vynásobí a následně sečte – výsledkem je tedy jedno číslo. Toto číslo udává, jak silně je ve vybrané části obrázku přítomen vzor, který tento filtr po natrénování CNN „reprezentuje“ (například hrana).

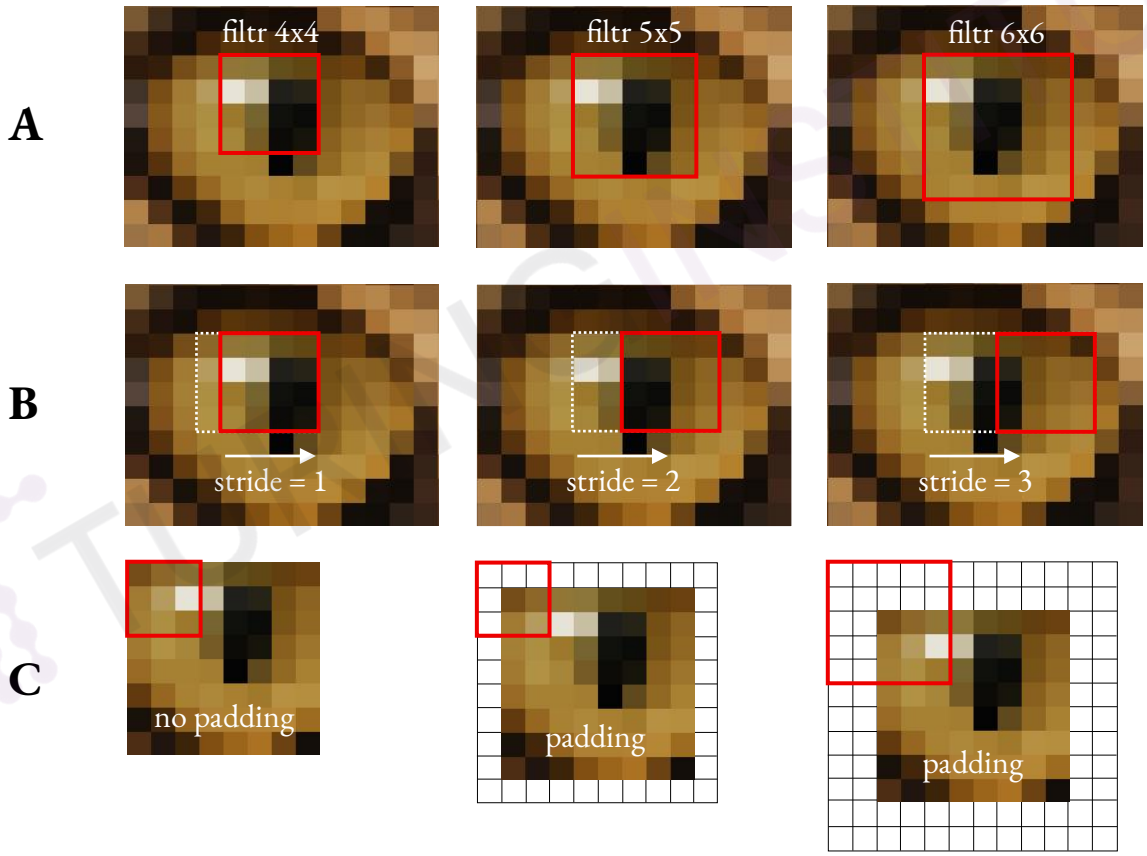
Tento postup se opakuje po celé ploše obrázku – filtr se posouvá krok za krokem (tento krok posunutí se nazývá „stride“ a musí se určit), až z operací vznikne nová mřížka čísel nazývaná „feature map“ (mapa příznaků). Tato mapa často bývá oproti původnímu obrázku redukována, protože filtr může nahradit větší oblast pixelů jedním číslem. To však neplatí vždy: výsledná velikost „feature“ mapy závisí totiž na použití tzv. „paddingu“, tedy na tom, zda je vstupní obrázek na okrajích rozšířen o dodatečné pixely, nebo ne. Pokud je „padding“ nulový, „feature“ mapa je skutečně menší než vstupní obrázek, pokud je však nenulový,

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Tento postup se opakuje po celé ploše obrázku – filtr se posouvá krok za krokem, přičemž velikost tohoto kroku se nazývá „stride“. Z těchto operací vzniká nová mřížka čísel zvaná „feature map“ (mapa příznaků). Výsledná velikost „feature“ mapy není pevně daná a závisí na třech parametrech:

- velikosti filtru (viz obrázek níže, část A),
- velikosti kroku posunu („stride“, viz obrázek níže, část B) a
- na tom, zda se při konvoluci používá rozšíření okrajů („padding“, viz obrázek níže, část C).

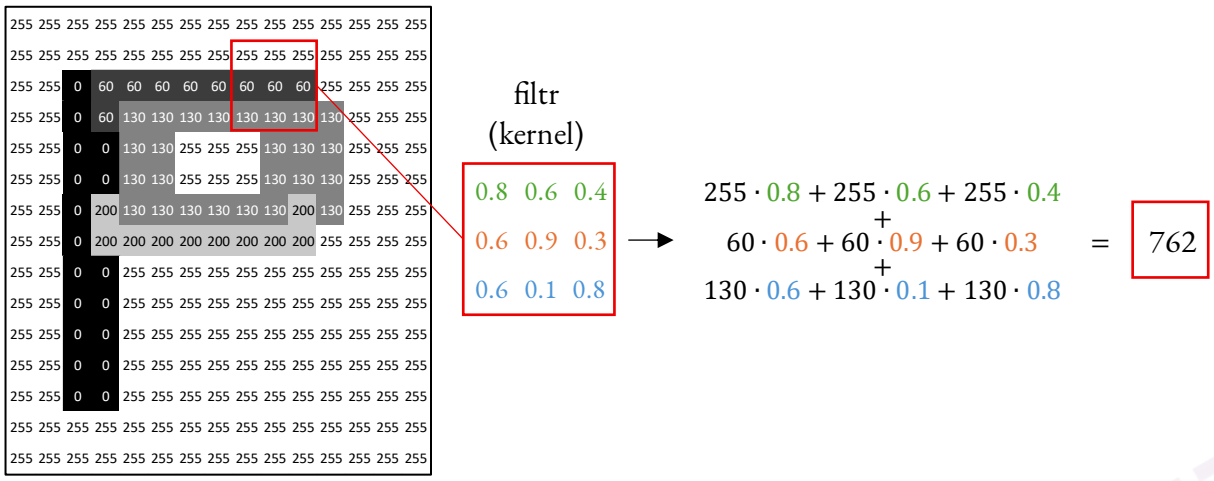
Bez funkce „padding“ je filtr aplikován pouze tam, kde kompletně „pasuje“ do obrázku, takže se ignorují krajní pixely – v tomto případě je výstup vždy menší než vstup. S funkcí „padding“ se kolem obrázku doplní umělé okraje (například nulové nebo zkopírované hodnoty), aby filtr mohl pokrýt i krajové oblasti. Tím lze zajistit, že „feature“ mapa bude mít stejné nebo téměř stejné rozměry jako původní obrázek. Hodnota „stride“ určuje, o kolik pixelů se filtr posune po každém kroku. Při „stride“ = 1 filtr postupuje po jednom pixelu a vytváří největší možný výstup. Větší „stride“ znamená větší skoky přes obraz, což vede k výraznému zmenšení výsledné „feature“ mapy. Celkově je velikost a obsah „feature“ mapy určena kombinací filtru, hodnoty „padding“ a „stride“, nikoli pouze samotným filtrem.



Konvoluční vrstvy tak pomáhají síti zachytit základní rysy obrázku, jako jsou hrany, jednoduché tvary nebo barevné přechody, které jsou pro rozpoznání obrázku zásadní.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Každý filtr se postupně umisťuje na různé části obrázku a vypočítává hodnotu, která udává, jak moc se obsah této malé části obrázku podobá. Funkce kernelu je znázorněna níže na obrázku.



Každý filtr (nebo také kernel) si můžeme představit jako malou tabulku s čísly, která postupně „přejíždí" přes celý obrázek. V každém místě se tato tabulka překryje přes odpovídající výřez pixelů a provede jednoduchý výpočet: každé číslo v tabulce se vynásobí s hodnotou pixelu, který leží přímo pod ním, a všechny tyto dílčí výsledky se nakonec sečtou do jednoho čísla. Výsledné číslo (například 762, viz obrázek) udává míru výraznosti určitého rysu. Čím vyšší, tím silněji filtr (kernel) v daném místě detekuje prvek, na který je vytrénovaný. Číslo se uloží do nové „feature" mapy přesně na pozici, která odpovídá středu aktuálního výřezu. Tímto způsobem se pixel po pixelu vykreslí celá „feature" mapa daného obrázku. Je velmi důležité si uvědomit, že čísla ve filtru (kernelu) nejsou pevné konstanty. Jsou to další trénovatelné parametry (váhy) sítě, které se optimalizují úplně stejně jako váhy v klasických dopředných neuronových sítích.

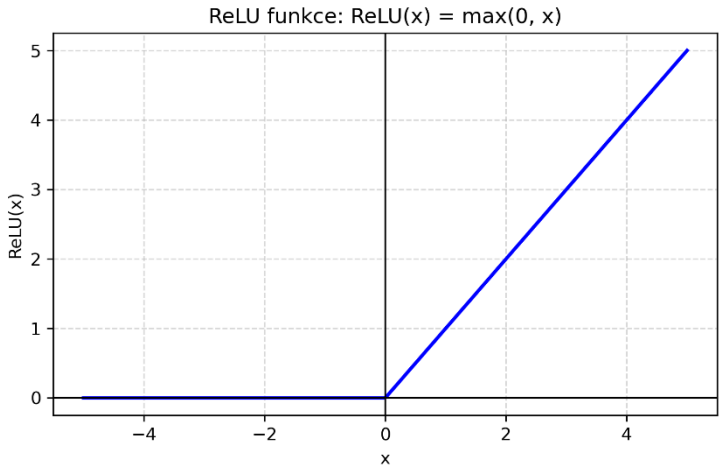
Poznámka k velikostem čísel výše v obrázku:

V reálných sítích se s takto vysokými čísly málokdy setkáme. Vstupní obrázky se předem normalizují (převedou na jednotné a menší měřítko, nejčastěji do rozsahu 0 až 1), takže i výsledky konvoluce jsou obvykle mnohem menší desetinná čísla. Pomáhá to síti učit se rychleji a stabilněji.

Aktivační funkce

Aktivační funkce se v CNN používají proto, aby síť dokázala pracovat i se složitými (nelineárními) vzory. Bez nich by byla schopná modelovat pouze velmi jednoduché vztahy, což by pro obrázky nestačilo. V konvolučních sítích se nejčastěji používá funkce ReLU („Rectified Linear Unit", viz obrázek níže). Ta zachovává pouze kladné hodnoty a záporné nahrazuje nulou. Díky tomu síť zvýrazní důležité signály a potlačí ty méně podstatné.

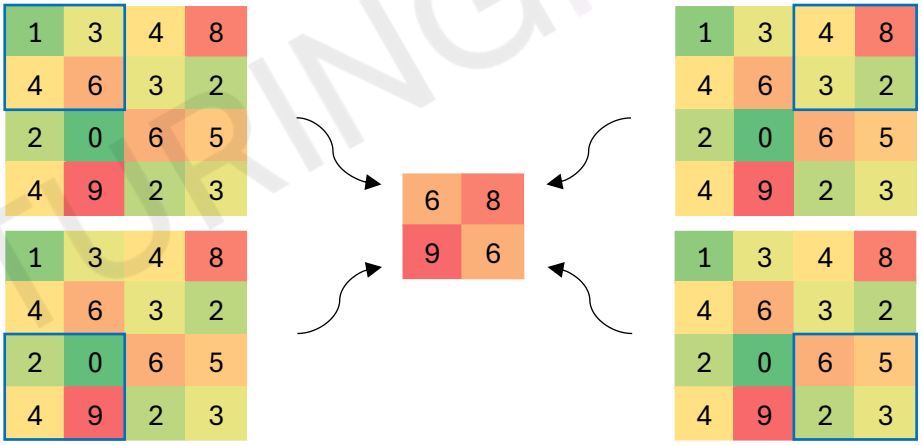
© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.



Pooling vrstvy

„Pooling“ je vrstva, která slouží ke zmenšení velikosti dat. To znamená, že síť po průchodu dat touto vrstvou pracuje s menším množstvím informací, což zvyšuje její rychlost a snižuje riziko jejího přeučení.

eNejčastěji se používá Max Pooling, který z malé oblasti obrázku (například z oblasti 2x2 pixelů, viz obrázek níže) vybere nejvyšší hodnotu. Tato vrstva tak z vybraných oblastí vždy vybere to nejdominantnější číslo, které má tuto oblast v dalších vrstvách sítě reprezentovat – vznikne tak nová mřížka čísel, která je oproti původní mřížce z předchozí vrstvy redukovaná. Díky tomu si síť uchová nejdůležitější informaci, ale odstraní drobné detaily, které nejsou pro rozpoznání obrázku zásadní.



„Pooling“ vrstva tak pomáhá zjednodušit data a zachovat pouze nejvýznamnější prvky.

Normalizační vrstvy

Normalizační vrstvy pomáhají udržet hodnoty v síti ve vhodném rozmezí. To vede ke stabilnějšímu a rychlejšímu trénování neuronové sítě. Jeden z druhů normalizace je například „batch normalization“.

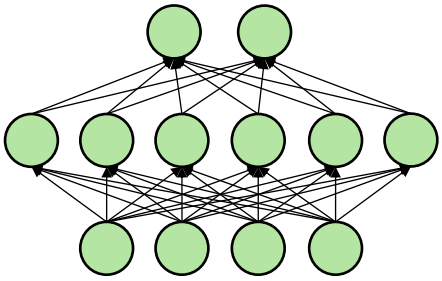
„Batch normalization“ je nejčastěji používaná normalizační vrstva. Upravuje hodnoty tak, aby jejich rozsah nebyl příliš velký nebo příliš malý, což zlepšuje chování a výkon celé sítě během jejího trénování.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

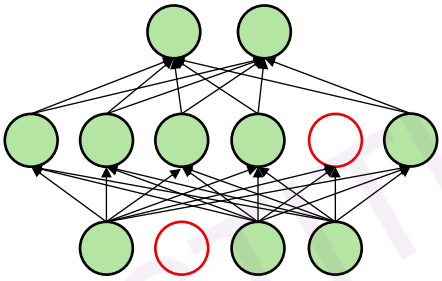
Dropout

„Dropout“ je vrstva, která slouží jako ochrana proti přeučení sítě. Pro připomenutí: přeučení znamená, že se síť naučí příliš přesně trénovací příklady a pak má problém správně vyhodnotit nové obrázky, které ještě neviděla.

Během trénování funkce „dropout“ náhodně vypíná určité procento neuronů (znemožňuje průchod dat těmito neurony), takže síť se nemůže spoléhat stále na stejné propojení a je „donucena“ hledat propojení nová, a tím se tak učit obecněji. Můžeme si to představit jako sportovní tým, ve kterém během tréninku občas náhodně chybí jeden z hráčů. Ostatní se musí naučit hrát dobře i bez něj, převzít jeho roli a spolupracovat jiným způsobem. Tím se tým stává pružnější a silnější. Stejně tak „dropout“ vede síť k tomu, aby se nespoléhala jen na několik neuronů, ale naučila se vytvářet více obecných a odolných řešení.



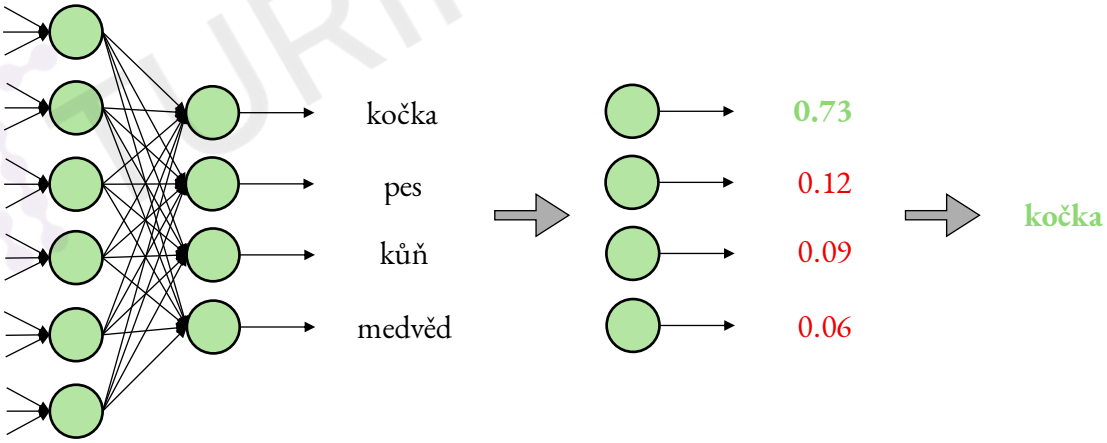
žádný dropout



dropout

Plně propojené vrstvy neuronů

Plně propojené vrstvy se obvykle nacházejí až na konci struktury CNN. Zatímco konvoluční a „pooling“ vrstvy z obrázku získávají a zpracovávají rysy, plně propojené vrstvy provádějí konečné rozhodnutí (viz obrázek níže).



Každý neuron v této vrstvě je propojen se všemi neurony z vrstvy předchozí. Síť tak kombinuje všechny získané informace a vytváří výstup – například rozhodne, zda je na obrázku kočka, pes, nebo auto.

V této části jsme se seznámili s nejdůležitějšími typy vrstev, které tvoří konvoluční neuronové sítě. Každá z nich plní vlastní úlohu – některé vrstvy z obrázku postupně získávají důležité rysy, jiné vrstvy data zjednodušují, pomáhají síti učit se stabilněji nebo zabraňují tomu, aby si příliš „zapamatovala“

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

trénovací příklady. Společně vytvářejí funkční celek, díky kterému dokáže CNN zpracovat obrázek a připravit si z něj podklady pro konečné rozhodnutí.

V následující kapitole si ukážeme, jak tyto vrstvy spolupracují v praxi. Projdeme si celý postup krok za krokem – od chvíle, kdy do sítě vstoupí obrázek, přes jednotlivé kroky až po závěrečné rozpoznání toho, co obrázek obsahuje. Díky tomu získáte ucelenější a jasnější představu o tom, jak CNN pracuje s obrázkem od začátku až do úplného konce.

1.5 Jak CNN krok za krokem zpracuje obrázek?

Nyní, když už známe základní stavební části konvoluční neuronové sítě, můžeme si ukázat, jak spolu jednotlivé vrstvy spolupracují při zpracování obrázku. Cílem této kapitoly je pochopit, co se s obrázkem v CNN děje od jeho vstupu až po konečné rozhodnutí – například že se na něm nachází kočka.

Aby byl proces co nejjasnější, projdeme si ho krok za krokem. U každého kroku je také připraven prostor pro jeho krátkou praktickou ukázku v programovacím jazyku Python s využitím knihovny PyTorch.

Krok 1: Vstupní obrázek vstoupí do sítě

Obrázek se nejprve načte a převede do číselné podoby (viz kapitola 1.3). Většina knihoven pracuje s obrázkem ve tvaru:

[batch, kanály, výška, šířka]

např.: [1, 3, 128, 128] = 1 obrázek, 3 barevné kanály (R, G, B), 128×128 pixelů

Ukázka v Pythonu:

```
from PIL import Image
import numpy as np
import torch
import torch.nn as nn

# 1) Načtení obrázku
img = Image.open("input.jpg").convert("RGB")
img = img.resize((128, 128))
img_np = np.array(img)

print("Tvar po načtení (H, W, C):", img_np.shape)

# 2) Převod na float a normalizace
img_np = img_np.astype("float32") / 255.0

# 3) Přeskládání pořadí os
# z (Height, Weight, Channel) → (Channel, Height, Weight)
img_chw = np.transpose(img_np, (2, 0, 1))
print("Tvar po transpozici (Channel, Height, Weight):", img_chw.shape)

# 4) Vytvoření batch tenzoru
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```
# z (C, H, W) → (1, C, H, W)
img_tensor = torch.tensor(img_chw).unsqueeze(0)
print("Tvar tensoru (Batch, Channel, Height, Weight):", img_tensor.shape)
```

Krok 2: První konvoluční vrstvy hledají základní rysy

První konvoluční vrstva se „dívá“ na malé části obrázku pomocí filtrů (např. 3x3 pixely) a může se v něm naučit hledat základní vizuální rysy, jako jsou:

- hrany (svislé, vodorovné, šikmé),
- jednoduché křivky a linie,
- barevné přechody.

Každý filtr vytváří vlastní „feature map“, která ukazuje, kde se daný rys v obrázku nachází. Po konvoluční vrstvě se obvykle použije aktivační funkce ReLU – funkce, která ponechá pouze kladné hodnoty a zbytek vynuluje. Tím zvýrazní výrazné rysy a potlačí nepotřebné informace.

Příklad změny tvaru dat po této části může být:

Vstup:	[1, 3, 128, 128]
Conv + ReLU:	[1, 8, 126, 126] (např. 8 filtrů 3×3)

Ukázka v Pythonu:

```
# 5) Definice první konvoluční vrstvy
# 8 filtrů o velikosti 3x3, stride=1, bez paddingu
conv1 = nn.Conv2d(
    in_channels=3,
    out_channels=8,
    kernel_size=3,
    stride=1,
    padding=0
)

# 6) Aplikace konvoluce
conv_out = conv1(img_tensor)
print("Tvar po konvoluci (B, C, H, W):", conv_out.shape)

# 7) Aplikace aktivační funkce ReLU
relu = nn.ReLU()
conv_out_relu = relu(conv_out)
print("Tvar po ReLU (B, C, H, W):", conv_out_relu.shape)
```

Krok 3: Pooling, který zjednodušuje informace

Poolingová vrstva zmenší rozměry dat a zachová nejdůležitější prvky. Nejčastěji se používá tzv. „max pooling“ – z malé oblasti (např. 2x2) vybere nejvyšší hodnotu. Efekt této vrstvy je následující:

- zrychluje práci sítě,
- snižuje počet parametrů a riziko přeučení,

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

– zachová nejvýraznější informace.

Příklad změny tvaru dat:

Před poolingem:	[1, 8, 126, 126]
Po poolingu:	[1, 8, 63, 63]

Ukázka v Pythonu:

```
# 8) Definice max pooling vrstvy
# 2x2 okno, stride = 2
maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

# 9) Aplikace poolingu
pool_out = maxpool(conv_out_relu)
print("Tvar po max poolingu (B, C, H, W):", pool_out.shape)
```

Krok 4: Další vrstvy hledají složitější struktury

Čím hlouběji v síti postupujeme, tím složitější souvislosti v datech dokáže model zachytit. I když se v jádru jedná o čistě matematické výpočty, které jsou pro lidskou mysl těžko uchopitelné, můžeme si tento proces pro lepší pochopení představit jako postupné skládání mozaiky:

– propojování detailů: prvotní linky a hrany se v dalších vrstvách začínají spojovat do konkrétnějších tvarů, jako jsou oblouky, mřížky nebo specifické barevné přechody,
– vnímání celkového obrazu: Tyto útvary se nakonec spojí do komplexních vzorů, které síti napovídají, co se na snímku nachází. Z dřívějších křivek tak vznikne celkový obrys objektu nebo jeho typická struktura.

Tvar dat se při dalším průchodu může dále měnit, například:

Po další Conv + ReLU:	[1, 16, 61, 61]
Po dalším Poolingu:	[1, 16, 30, 30]

V této fázi síť už nepracuje se separátními pixely samostatně, ale spíše s informacemi o obrázku.

Ukázka v Pythonu:

```
# 10) Druhá konvoluční vrstva
# Vstupních kanálů = 8 (z předchozí vrstvy)
# Výstupních kanálů = 16
conv2 = nn.Conv2d(
    in_channels=8,
    out_channels=16,
    kernel_size=3,
    stride=1,
    padding=0
)
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```python
# 11) Aplikace druhé konvoluce
conv2_out = conv2(pool_out)
print("Tvar po 2. konvoluci (B, C, H, W):", conv2_out.shape)

# 12) ReLU aktivace
conv2_out_relu = relu(conv2_out)
print("Tvar po ReLU (B, C, H, W):", conv2_out_relu.shape)

# 13) Druhý pooling
maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2)
pool2_out = maxpool2(conv2_out_relu)
print("Tvar po 2. poolingu (B, C, H, W):", pool2_out.shape)
```

Krok 5: Převod dat na konečné rozhodnutí

Po posledním „poolingu“ jsou data ještě stále ve formě „feature maps“ (map rysů), ne ve formě seznamu hodnot. Proto je nejprve musíme převést na jednorozměrný vektor – samostatný řádek či sloupec čísel – tomuto kroku říkáme **flattening**.

Příklad změny tvaru:

Před flatteningem:	[1, 16, 30, 30]
Po flatteningu:	[1, 14400]

Tento vektor (řada čísel) se poté předá plně propojené vrstvě, která:

- spojí veškeré získané informace,
- vyhodnotí, co obrázek nejspíše obsahuje,
- vrátí „skóre“ nebo pravděpodobnosti pro všechny možné třídy (třída reprezentuje označení obsahu na obrázku – pes, kočka, auto apod.).

Ukázka v Pythonu:

```python
# 14) Flattening – převod feature maps na jednorozměrný vektor
# z (B, C, H, W) → (B, C*H*W)
batch_size = pool2_out.size(0)
flat = pool2_out.view(batch_size, -1)
print("Tvar po flatteningu (B, N):", flat.shape)

# 15) Plně propojená (fully connected) vrstva
# zde příklad pro 4 třídy (např. pes, kočka, kůň, medvěd)
fc = nn.Linear(
    in_features=flat.size(1),
    out_features=4
)

# 16) Výpočet výstupních skóre (logitů) pro jednotlivé třídy
logits = fc(flat)
print("Tvar výstupu plně propojené vrstvy (B, num_classes):", logits.shape)
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```python
# 17) Převod na pravděpodobnosti pomocí softmax
softmax = nn.Softmax(dim=1)
probs = softmax(logits)
print("Pravděpodobnosti tříd:", probs)
```

Shrnutí celého procesu:

1. Obrázek vstoupí do CNN a převede se do číselné podoby.
2. Konvoluční vrstvy + ReLU vyhledávají rysy (hrany, tvary a pak také části objektů).
3. Pooling vrstva data zjednodušuje a odstraňuje nepodstatné detaily.
4. Hlubší vrstvy skládají rysy do složitějších struktur.
5. Flattening a plně propojená vrstva určí, co na obrázku je.

Tento proces umožňuje CNN chápat obrázek postupně – od drobných detailů až po celek – podobně jako to dělá lidské oko a mozek.

1.6 Jak se CNN naučí rozpoznat obrázek?

V předchozí kapitole jsme si ukázali, jak konvoluční neuronová síť zpracovává obrázek krok za krokem. Aby ale síť dokázala správně rozpoznat, co se na obrázku nachází, musí se to nejprve naučit. V této kapitole si vysvětlíme, jak proces učení probíhá.

Na první pohled může působit téměř magicky, že síť po čase sama pozná kočku, psa nebo auto. Ve skutečnosti však proces učení vychází z poměrně jednoduché myšlenky: síť postupně upravuje své vnitřní hodnoty (parametry, zejména váhy filtrů), aby se její výstup co nejvíce přiblížil správnému výsledku.

Celý proces učení CNN lze rozdělit do několika kroků:

1. Síť dostane trénovací obrázek a správný výsledek – čili skutečnost, jestli se na obrázku nachází kočka, pes nebo například něco jiného. Takové obrázky tvoří tzv. trénovací data.

   Každý obrázek:
   - projde sítí,
   - síť vytvoří odhad (například na 70 % se na obrázku nachází kočka, na 25 % pes a na 5 % auto),
   - tento odhad porovnáme se správnou odpovědí.

2. Vyhodnotí se chyba („loss“)

   Po porovnání odhadu konvoluční sítě s realitou (co se na obrázku opravdu nachází) vypočítáme chybu modelu, často nazývanou „loss“ (hodnota ztrátové funkce). Čím větší je rozdíl mezi odhadem a správnou odpovědí, tím je chyba modelu „loss“ vyšší.

   Zjednodušeně:

Síť řekla:	0.42 (pes)
Správná odpověď byla:	1.00 (kočka)
Síť se mýlila, což znamená vysoká chyba („loss“).	

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

3. Síť upraví své vnitřní parametry (váhy aj.)

   Aby se síť postupně zlepšovala, musí upravovat své parametry – především:

   - hodnoty filtrů v konvolučních vrstvách,
   - váhy v plně propojených vrstvách.

   Hodnoty těchto parametrů určují, jakým způsobem síť zpracovává obrázky. K jejich úpravě se používá algoritmus zvaný zpětné šíření chyby („loss backpropagation“), který spolupracuje s optimalizačním algoritmem, jenž rozhoduje o tom, jak moc mají být jednotlivé parametry upraveny, aby se chyba predikce konvoluční neuronové sítě zmenšovala.

4. Opakování výše uvedených bodů

   Učení jedním obrázkem nestačí – síť musí vidět tisíce až miliony příkladů. Po každé dávce („batch“) těchto obrázku síť při svém tréninku upraví své váhy směrem vedoucím ke snížení celkové chyby. Postupným opakováním se její schopnost rozpoznávat objekty na obrázcích stále zlepšuje.

   V průběhu tréninku dochází k tomu, že:

   - nízké vrstvy se naučí hledat základní vzory,
   - střední vrstvy skládají tyto prvky dohromady do částí objektů a
   - vyšší vrstvy rozpoznávají celé objekty.

   Konvoluční neuronová síť tedy nezačne rozumět obrázkům hned, ale učí se postupně.

5. Testování toho, zda se síť skutečně něco naučila

   Síť v posledním bodě testujeme na obrázcích, které při tréninku nikdy neviděla. Pokud správně rozpozná i nové obrázky, můžeme konstatovat, že se síť kvalitně naučila a nedošlo k přeučení nebo k nedostatečnému naučení. Pokud síť správně funguje pouze na obrázcích, které viděla již při tréninku (trénovací data), ale selhává na nových, je přeučená, pokud není síť schopna rozpoznat ani obrázky z trénovacích dat, je nedostatečně naučená. I zde vidíme klasické rozdělení dat při tréninku neuronových sítí na:

   - trénovací data – síť se z nich učí,
   - validační data – přesnost sítě se na nich vyhodnocuje během tréninku, abychom v reálném čase viděli, jestli jde trénink správným směrem, či nikoliv,
   - testovací data – síť na nich provádí finální vyhodnocení své přesnosti po dokončení jejího tréninku.

<u>Shrnutí celého procesu:</u>

Konvoluční neuronová síť se učí tak, že:

1. dostane obrázek a vytvoří odhad toho, jaký objekt se na něm nachází,
2. porovná ho se správnou odpovědí a vypočítá chybu („loss“),
3. pomocí zpětného šíření chyby a optimalizačního algoritmu upraví své vnitřní parametry,
4. tento proces následně opakuje tisíckrát až milionkrát,
5. postupně se při správném nastavení zlepšuje, až zvládá rozpoznávat objekty s vysokou přesností.

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

1.7 Kde všude se CNN používají a jak se k nim dostat

Konvoluční neuronové sítě dnes patří mezi nejpoužívanější nástroje pro zpracovávání obrazových dat a v některých případech významně ovlivňují náš každodenní život – často i v situacích, kdy si to ani neuvědomujeme. V této kapitole si ukážeme, kde se s CNN můžeme setkat v praxi, a naznačíme, jak s nimi může začít pracovat i začátečník.

CNN v praxi – kde se s nimi setkáváme?

1. Rozpoznávání objektů na obrázcích a ve videu

   CNN se používají k tomu, aby určily, co se nachází na obrázku nebo v jednotlivých snímcích videa, například:

   - rozpoznávání obličejů v mobilních telefonech,
   - třídění fotografií podle obsahu (kočka, pes, auto aj.),
   - filtrování nevhodného obsahu na sociálních sítích apod.

2. Autonomní řízení a dopravní systémy

   Autonomní auta využívají CNN ke zpracování obrazu z kamer, aby dokázala vidět okolní svět, například:

   - rozpoznávání dopravních značek,
   - detekce chodců, aut nebo překážek,
   - sledování jízdních pruhů.

3. Zdravotnictví a lékařská diagnostika

   CNN pomáhají lékařům analyzovat snímky z medicíny. Dokážou si všimnout detailů, které mohou být pro lidské oko obtížně postřehnutelné, například:

   - analýza rentgenů, magnetických rezonancí či CT snímků,
   - včasná detekce nádorů nebo abnormalit,
   - asistence při diagnostice různých nemocí apod.

4. Bezpečnost a dohledové systémy

   Využívají se v kamerových systémech k detekci pohybu, rozpoznání osob nebo podezřelého chování.

5. Další oblasti použití, jako například kontrola kvality výrobků (odhalování vad na součástkách aj.), nebo například i v zábavních sférách (využití v počítačových hrách) aj.

Jak se k práci s CNN může dostat i začátečník?

I když mohou CNN působit složitě, dnes je možné s nimi začít pracovat mnohem snadněji než dříve. Stačí postupovat po malých krůčcích a využít dostupné nástroje a zdroje.

1. Praktické prostředí, kde lze CNN vyzkoušet

   Denně používané knihovny a platformy, které jsou vhodné pro začátečníky:

   - Google Colab – nic se nemusí instalovat, běží v prohlížeči,
   - Jupyter Notebook – ideální forma, pokud chceme programovat lokálně na svém PC,
   - PyTorch, TensorFlow, Keras – nejpoužívanější knihovny pro hluboké učení.

2. Doporučený postup pro první experimenty s CNN

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Pokud chcete začít od jednoduchých příkladů, vhodná cesta může být:

a. porozumět základům neuronových sítí – co je neuron, váhy, aktivační funkce, trénování apod.
b. vyzkoušet si první malou CNN – využít jednu z dříve uvedených knihoven s pomocí například nějakého návodu (velké množství videí na internetu) a použití jednoduché CNN například na datasetu ručně psaných číslic MNIST.
c. naučit se zobrazit mezivýstupy konvolučních vrstev (tzv. „feature maps“). Díky tomu uvidíte, jaké rysy obrázku síť v jednotlivých vrstvách „vnímá“. V praxi to zvládne i začátečník pomocí několika řádků kódu.
d. vyzkoušet složitější dataset (například zvířata aj.),
e. začít experimentovat – měnit počet vrstev, filtry, pooling vrstvy, dropout vrstvy a sledovat, jak se mění výsledky – pomůže k pochopení chování neuronové sítě při změně jejích hyperparametrů.

3. Zdroje, které mohou při učení pomoci:
   - krátké online kurzy a tutoriály,
   - otevřené datové sady (MNIST, CIFAR, ImageNet aj.),
   - GitHub projekty, ukázkové kódy, Kaggle ad.,

Shrnutí:

Konvoluční neuronové sítě mají dnes obrovské využití napříč moderním světem – od rozpoznávání obličejů v telefonu přes medicínu až po řízení aut. Přestože mohou na první pohled působit složitě, existuje mnoho jednoduchých cest a nástrojů, které začátečníkům umožňují CNN vyzkoušet, pochopit a později s nimi pracovat na vlastních projektech.

1.8 Praktické cvičení s řešením

Níže najdeš jednoduché, ale kompletní praktické cvičení: postavíme a natrénujeme malou CNN, která rozpoznává ručně psané číslice z datasetu MNIST. Na konci navíc vizualizujeme feature maps z první konvoluční vrstvy, aby bylo vidět, „co síť vnímá“.

Zadání:

Cíl: Naučit jednoduchou CNN rozpoznat číslice (0-9) z obrázků o velikosti 28x28 pixelů.
Data: MNIST (60000 trénovacích a 10000 testovacích obrázků).
Postup: Načíst data – normalizovat – definovat CNN – natrénovat (pár epoch) – ověřit přesnost – zobrazit „feature maps“ pro 1 ukázkový obrázek.

Kompletní řešení v Pythonu s využitím knihovny PyTorch:

```
# =========================
# 1) KNIHOVNY & NASTAVENÍ
# =========================
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```
import numpy as np
import random
import os

# Reproducibilita (volitelné)
seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Zařízení:", device)

# =========================
# 2) DATA: MNIST
# =========================
# Transformace: převod na Tensor a normalizace do rozumného rozsahu
transform = transforms.Compose([
    transforms.ToTensor(),                # [0,1]
    transforms.Normalize((0.1307,), (0.3081,))  # standardní normalizace pro
MNIST
])

train_ds = datasets.MNIST(root="./data", train=True,  download=True,
transform=transform)
test_ds  = datasets.MNIST(root="./data", train=False, download=True,
transform=transform)

train_loader = DataLoader(train_ds, batch_size=64,
shuffle=True,  num_workers=2, pin_memory=True if device.type=="cuda" else
False)
test_loader  = DataLoader(test_ds,  batch_size=256, shuffle=False,
num_workers=2, pin_memory=True if device.type=="cuda" else False)

# =========================
# 3) MODEL: malá konvoluční síť
# =========================
class SmallCNN(nn.Module):
    def __init__(self):
        super().__init__()
        # Vstup je 1×28×28 (MNIST je černobílý)
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3,
padding=1)  # -> 8×28×28
        self.conv2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,
padding=1) # -> 16×14×14 (po poolingu)
        self.pool  = nn.MaxPool2d(2)  # půlí výšku i šířku: 28->14->7

        # Po dvou poolinzích: 16 kanálů, prostor 7×7 => 16*7*7
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```
        self.fc1   = nn.Linear(16*7*7, 64)
        self.fc2   = nn.Linear(64, 10)  # 10 tříd (0–9)

    def forward(self, x):
        # Blok 1: Conv->ReLU->Pool
        x = self.conv1(x)           # 8×28×28
        x = F.relu(x)
        x = self.pool(x)            # 8×14×14

        # Blok 2: Conv->ReLU->Pool
        x = self.conv2(x)           # 16×14×14
        x = F.relu(x)
        x = self.pool(x)            # 16×7×7

        # Flatten -> Dense -> ReLU -> Dense
        x = torch.flatten(x, 1)     # [batch, 16*7*7]
        x = F.relu(self.fc1(x))
        x = self.fc2(x)             # logits (nesoftmaxované skóre pro 10
tříd)
        return x

model = SmallCNN().to(device)
print(model)

# =========================
# 4) OPTIMALIZACE
# =========================
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

# =========================
# 5) TRÉNINK & VALIDACE
# =========================
def train_one_epoch(epoch):
    model.train()
    running_loss, correct, total = 0.0, 0, 0
    for imgs, labels in train_loader:
        imgs, labels = imgs.to(device), labels.to(device)

        optimizer.zero_grad()
        logits = model(imgs)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * imgs.size(0)
        preds = logits.argmax(dim=1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```
    print(f"[E{epoch}] train loss: {running_loss/total:.4f} | train acc:
{correct/total*100:.2f}%")

@torch.no_grad()
def evaluate():
    model.eval()
    correct, total = 0, 0
    for imgs, labels in test_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        logits = model(imgs)
        preds = logits.argmax(dim=1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
    acc = correct/total*100
    print(f"test acc: {acc:.2f}%")
    return acc

EPOCHS = 3  # krátký trénink pro demo; můžeš zvýšit pro lepší přesnost
for epoch in range(1, EPOCHS+1):
    train_one_epoch(epoch)
    evaluate()

# =========================
# 6) VIZUALIZACE FEATURE MAPS (mezivýstupů)
# =========================
# Vezmeme 1 testovací obrázek, pustíme ho přes první konvoluční vrstvu
# a vykreslíme několik map rysů, aby bylo vidět, co vrstva „vnímá“.
model.eval()

# Vezmeme 1 batch z test_loaderu
images, labels = next(iter(test_loader))
images = images.to(device)

# Pro feature maps použijeme "hook" nebo prostě zavoláme dílčí průchod
# Zde přímo zavoláme conv1 a ReLU+pool, abychom viděli výstup po první
konvoluci.
with torch.no_grad():
    x = model.conv1(images)   # tvar: [batch, 8, 28, 28]
    x = F.relu(x)

# Vybereme první obrázek v batchi a jeho mapy rysů (kanály)
fmaps = x[0].detach().cpu()  # tvar: [8, 28, 28] – 8 feature maps

# Vykreslíme prvních 8 map (nebo méně, podle out_channels)
n_show = min(8, fmaps.shape[0])
cols = 4
rows = int(np.ceil(n_show/cols))
```


© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

```
plt.figure(figsize=(8, 2*rows))
for i in range(n_show):
    plt.subplot(rows, cols, i+1)
    plt.imshow(fmaps[i].numpy(), cmap='gray')
    plt.axis('off')
    plt.title(f"Map {i}")
plt.suptitle("Feature maps – výstup z 1. konvoluční vrstvy (po ReLU)", y=1.02)
plt.tight_layout()
plt.show()

# ========================
# 7) PŘEDPOVĚĎ NA NĚKOLIKA OBRÁZCÍCH
# ========================
@torch.no_grad()
def show_preds(n=8):
    model.eval()
    images, labels = next(iter(test_loader))
    images, labels = images.to(device), labels.to(device)
    logits = model(images)
    preds = logits.argmax(dim=1)

    plt.figure(figsize=(12, 3))
    for i in range(n):
        plt.subplot(1, n, i+1)
        plt.imshow(images[i].cpu().squeeze(), cmap='gray')
        plt.title(f"pred: {preds[i].item()}\ntrue: {labels[i].item()}")
        plt.axis('off')
    plt.suptitle("Ukázkové předpovědi")
    plt.tight_layout()
    plt.show()

show_preds(8)
```

TURINGINSTITUT

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.

Příloha – více matematiky

??? Chceme?

TURINGINSTITUT

© Bee Learning s.r.o., IČO 10827161. Licence Turing Institut (turinginstitut.cz): materiál smí být použit pouze ve výuce Turing Institut. Komerční užití a použití k jakékoli jiné výuce je zakázáno.