



Vilniaus universitetas
Matematikos ir informatikos fakultetas
Informatikos katedra



Konvoliuciniai neuroniniai tinklai

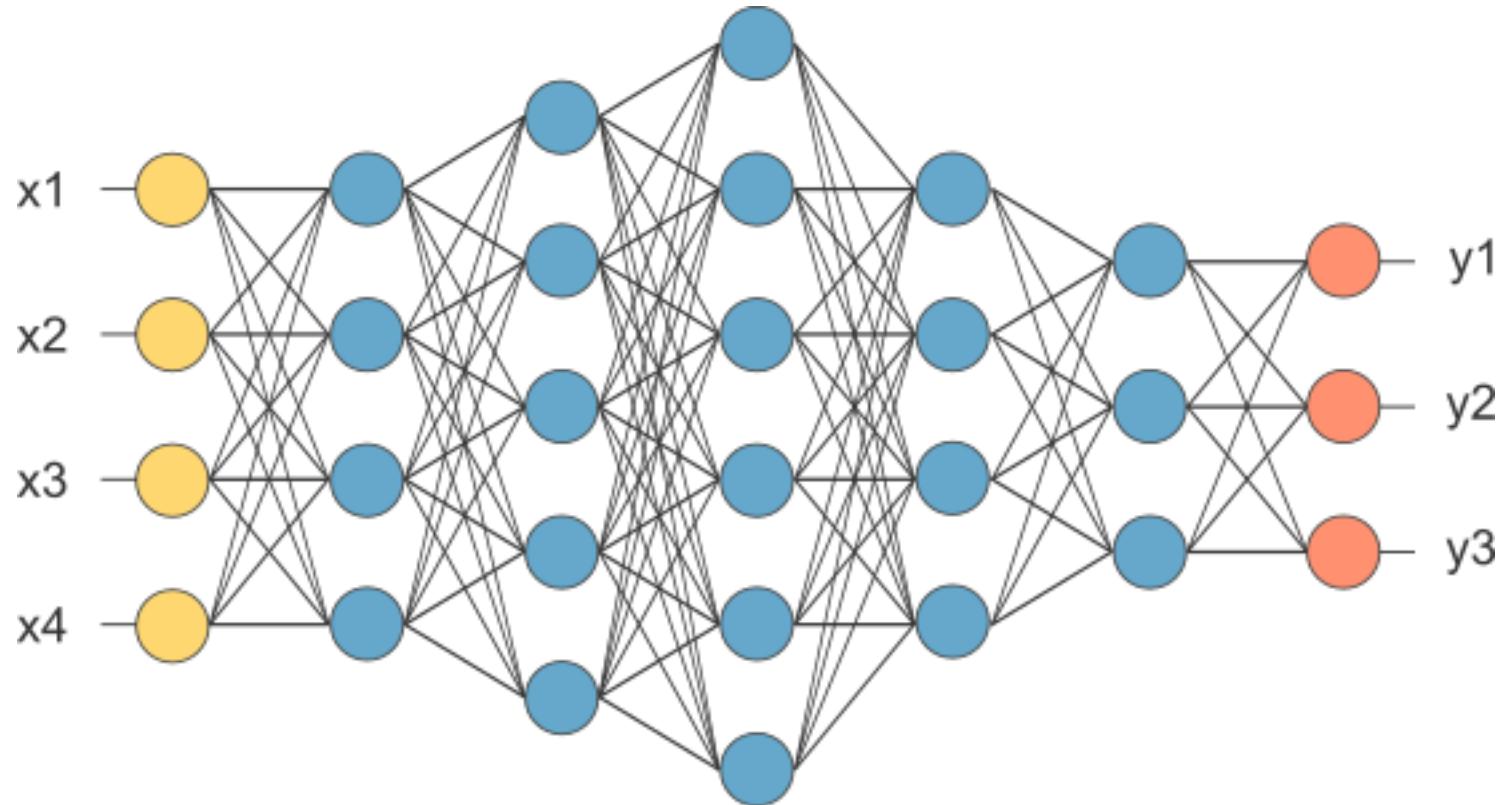
prof. dr. Olga Kurasova
Olga.Kurasova@mii.vu.lt

2018

Gilioji neuroniniai tinklai

- Nors **tiesioginio sklidimo neuroniniai tinklai** gali turėti norimą kiekį paslėptų sluoksnių ir neuronų skaičių juose, tačiau dėl buvusio skaičiavimų resursų ribotumo, dažnai buvo naudojami **tik vienas ar du paslėpti sluoksniai**, turintis po kelis neuronus.
- Šiuo metu plečiantis skaičiavimo resursų pajėgumams, atsirado galimybė naudoti **daugiau sluoksnių** ir **daugiau neuronų** juose.
- Taip išpopuliarejo **gilioji neuroniniai tinklai**.

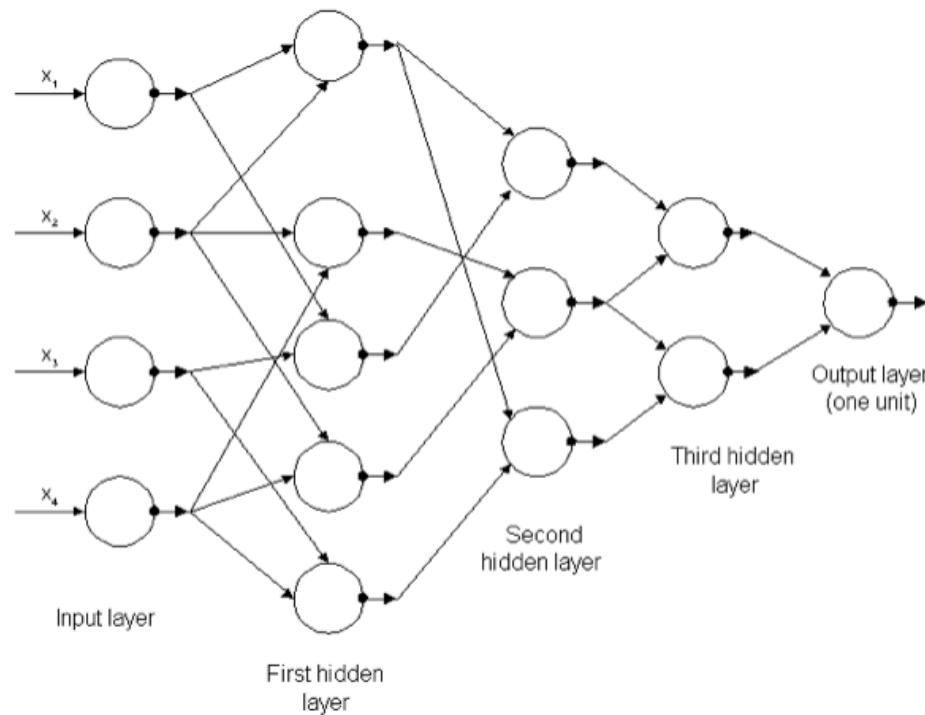
Giliųjų NN struktūra



Paveikslas iš <http://www.opennn.net>

Giliųjų DNT pradžia

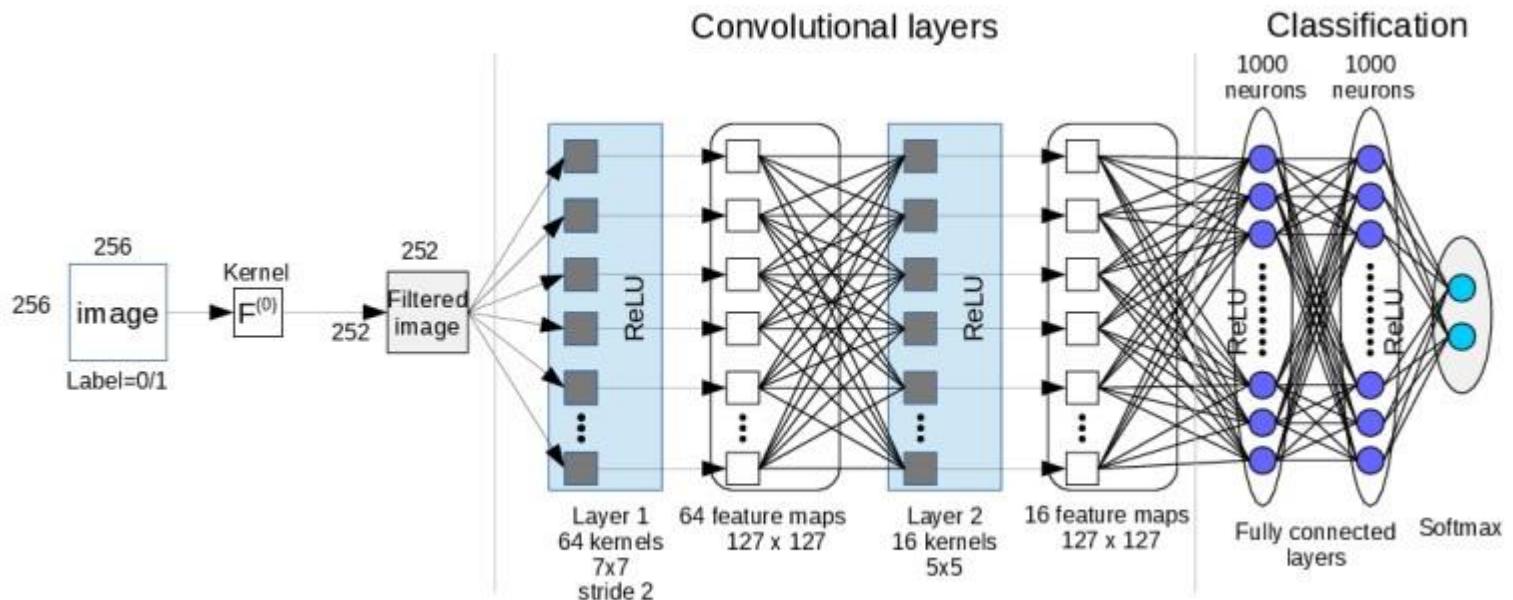
- **Giliųjų DNT pradžia** laikoma **1965** m.
- Ivakhnenko, A. G. and Lapa, V. G. (1965). Cybernetic Predicting Devices. CCM Information Corporation.



Konvoliuciniai neuroniniai tinklai

- **Konvoliuciniai neuroniniai tinklai** (angl. *convolution neural networks, CNN, ConvNet*) – tai vieni populiariausių giliųjų neuroninių tinklų tipų.
- Dar vadinami **sąsukiniais** neuroniniais tinklais.
- Pradininkas **Yann LeCun** (dirbantis Facebook dirbtinio intelekto tyrimų grupėje) (https://en.wikipedia.org/wiki/Yann_LeCun), pirmasis panaudojo CNN **ranka rašytiems skaitmenims atpažinti** (MNIST duomenų aibė).

Konvoliuciniai neuroniniai tinklai



<https://medium.com/@eternalzer0dayx/demystifying-convolutional-neural-networks-ca17bdc75559>

Konvoliucija (sąsuka)

- CNN pavadinimas kilo nuo **matematinės operacijos** – konvoliucija (sąsuka).
- **Konvoliucija** – tai matematinė operacija, kuri paima dvi funkcijas f ir g ir grąžina trečią, kuri, tam tikra prasme, parodo f ir g persidengimo lygi.
- Dažniausiai viena funkcija imama kaip fiksuotas filtras, dar vadinamas **branduoliu** (angl. *kernel*).
- Funkcijų f ir g konvoliucija žymima $f * g$. Ji apibrėžiama kaip **funkcijų sandaugos integralas**, po to, kai viena jų buvo paversta ir padauginta iš -1 .

Konvoliucija (sąsuka)

- Konvoliucija yra **integralinės transformacijos rūšis**:

$$(f * g)(t) = \int_a^b f(\pi)g(t - \pi)d\pi$$

- Integravimo **rėžiai** priklauso nuo funkcijų apibrėžimo srities. Dažniausiai $a = -\infty$ ir $b = +\infty$.

Konvoluciijos savybės

- Komutatyvumas: $f * g = g * f$
- Asociatyvumas: $f * (g * h) = (f * g) * h$
- Distributyvumas: $f * (g + h) = (f * g) + (f * h)$
- Vienetinis elementas: $f * \delta = \delta * f = f$
- Daugybos su skaliaru asociatyvumas:

$$a(f * g) = (af) * g = f * (ag)$$

kiekvienam realiam skaičiui a .

Diskreti konvoliucija

- Iprastai mašininio (giliojo) mokymo atvejais, nagrinėjant duomenis, besikeičiančius laike, susiduriama su **diskrečiais dalykais**.
- Diskrečioms funkcijoms apibrėžiama **diskrečios konvoliucijos** operaciją:

$$s(t) = (f * g)(t) = \sum_{n=-\infty}^{\infty} f(n)g(t - n)$$

- Naudojant šią formulę **konvoliucijos sudėtingumas** yra lygus $O(N^2)$ aritmetinių operacijų N taškams. Tačiau šis dydis gali būti sumažintas iki $O(N \log N)$, panaudojant greitesnius algoritmus.

Konvoliucija vaizdams

- Turint dvimatį vaizdą I kaip įvestį, naudojant **dvimatį branduolių (filtrą) K** :

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

- **Konvoliucija** yra komutatyvi, tad:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

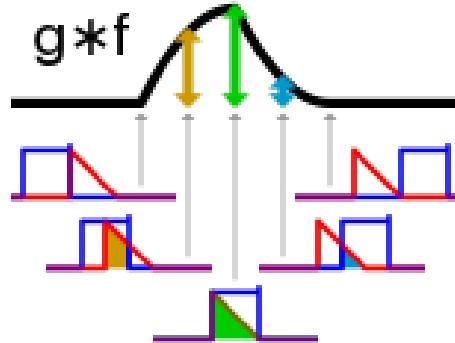
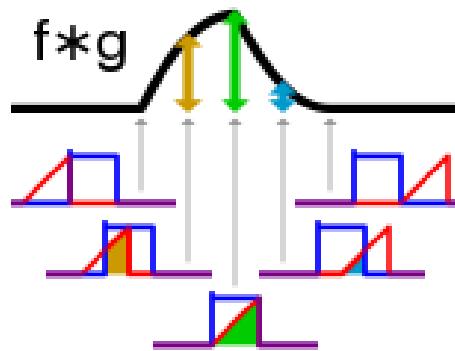
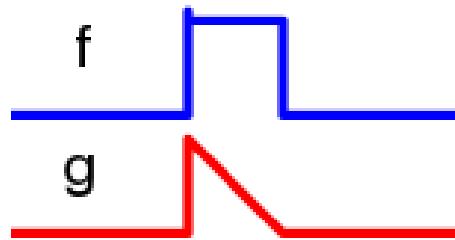
- Įprastai naudojama antroji formulė.

Konvoliucija vs kryžminė koreliacijos

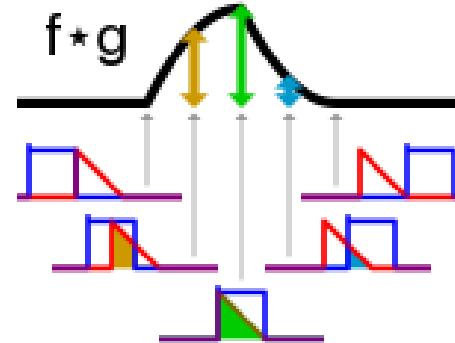
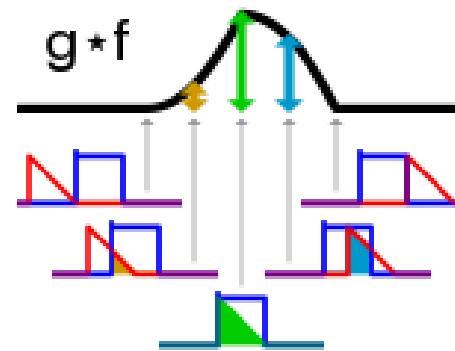
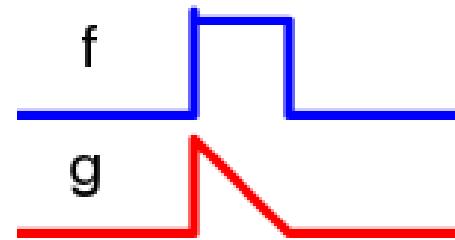
- Kadangi konvoliucija yra labai panaši į **kryžminę koreliaciją** (angl. *cross-correlation*), dažnai šie terminai vartojami kaip sinonimai.
- Kryžminės koreliacijos atveju nėra branduolio funkcijos apvertimo:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

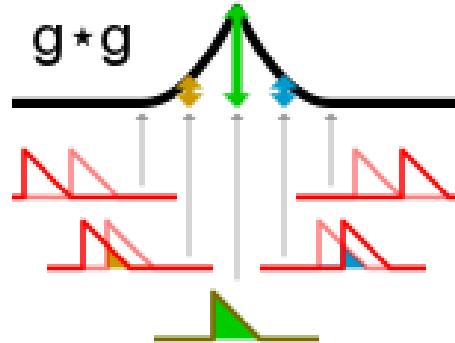
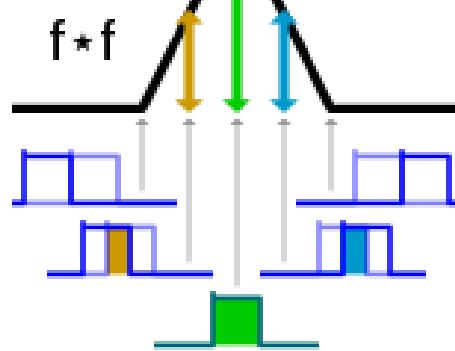
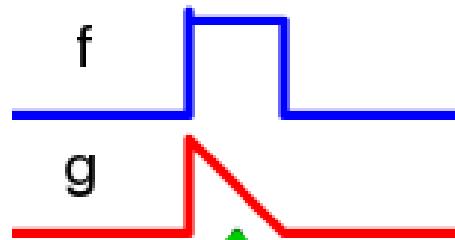
Convolution



Cross-correlation

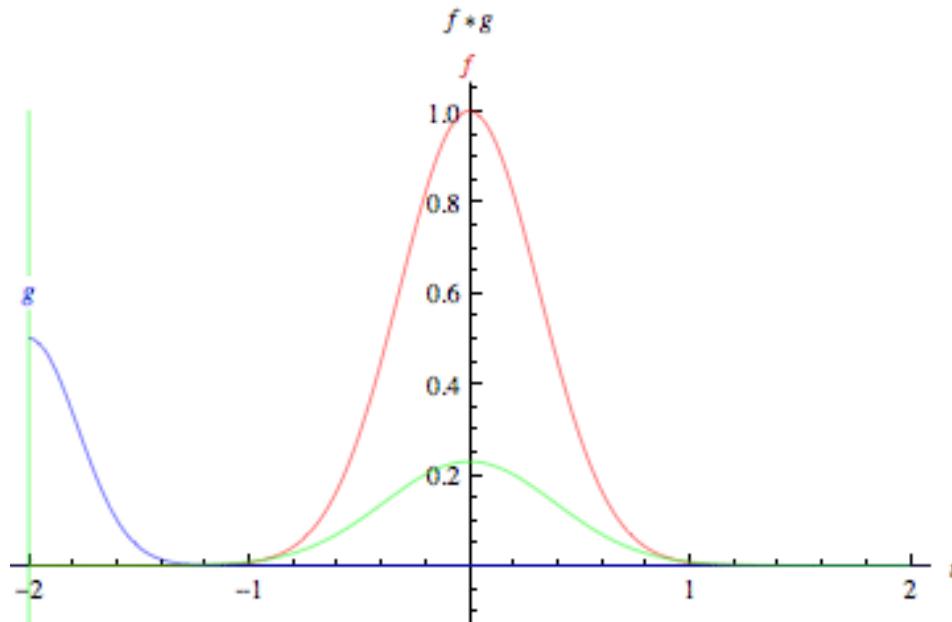


Autocorrelation



Konvoliucija

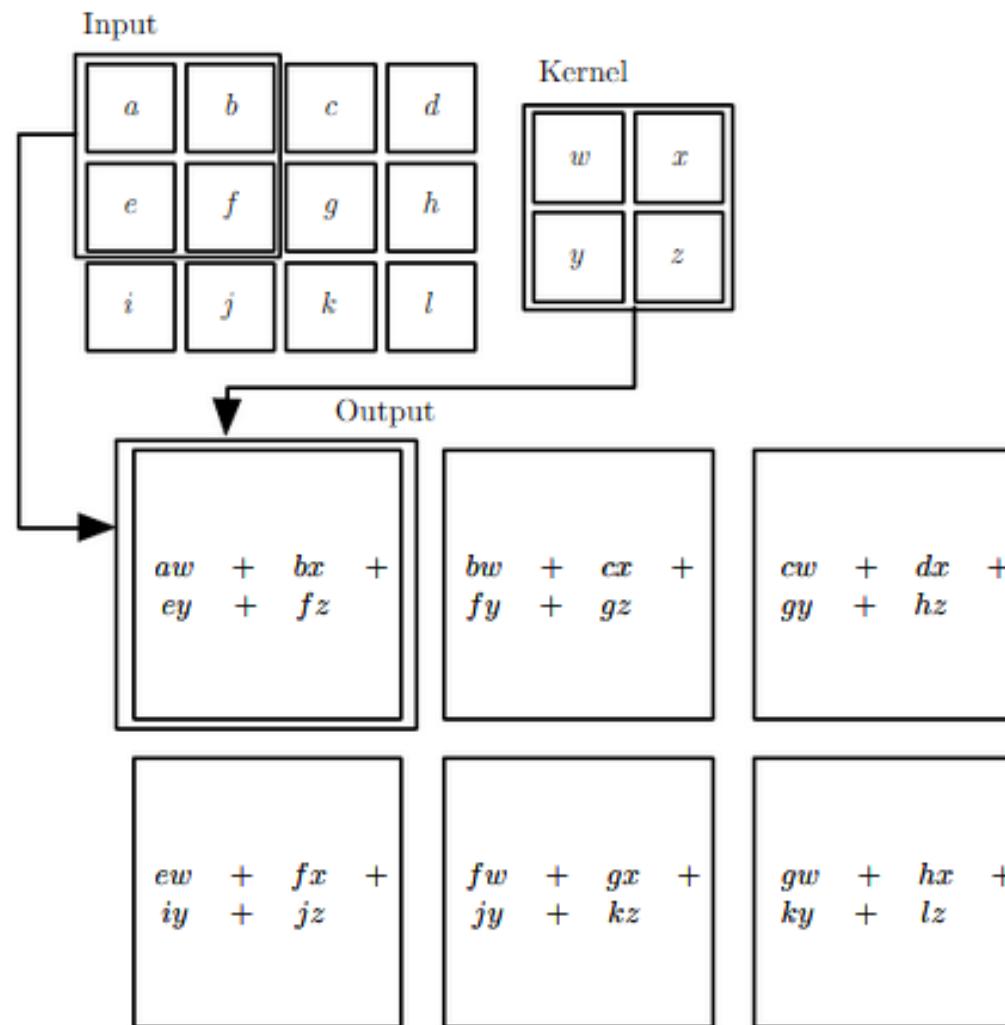
- Žalia kreivė parodo **mėlynos** ir **raudonos** kreivių konvoliuciją, t.y. šių funkcijų persidengimą.



Interaktyvų paveiksluką rasite:

<https://skymind.ai/wiki/convolutional-network>

Konvoliucijos pavyzdys (be branduolio apvertimo)



Konvoliucija matricos daugybos prasme

- Diskreti konvoliucija gali būti **išreiškiama daugyba iš matricos**, turinčios kelis elementus, kurie yra lygūs kitiems elementams.
- **Pavyzdžiu**, vienmatei diskrečiai konvoliucijai kiekviena matricos eilutė turi būti lygi ankstesnei eilutei, perustumtai per vieną elementą.

$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

Konvoliucija – išretinta matrica

Atsižvelgiant į tai, kad įprastai branduolio matrica yra daug mažesnė nei vaizdo matrica, be to, keli matricos elementai turi būti lygūs vienas kitam, **konvoliucija atitinka labai išretintą matricą** (angl. *sparse matrix*) – matricą, kurios didžioji dalis elementų lygūs 0.

Konvoliuciniai neuroniniai tinklai

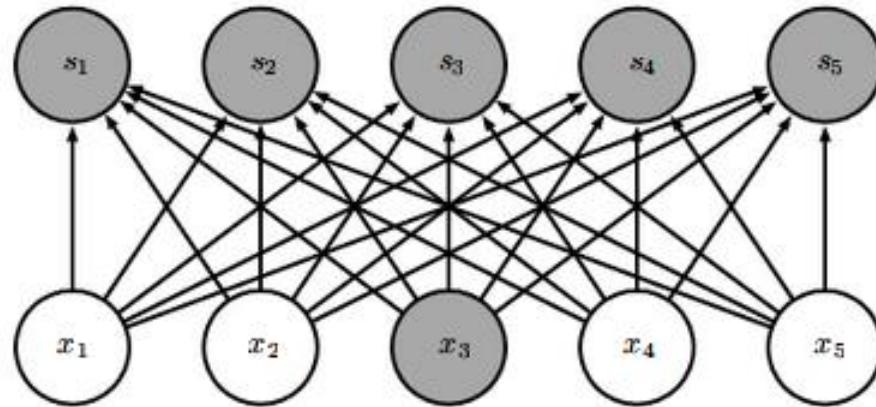
- **Tradiciniuose neuroniniuose tinkluose** naudojama matricos daugyba iš parametru matricos, kur parametrai nurodo sąveiką tarp įvesties ir išvesties. Tai reiškia, kad kiekviena išvestis sąveikauja su kiekvienu įėjimu.
- **Konvoliuciniai tinklai** yra paprasčiausiai neuroniniai tinklai, naudojantys konvoluciujos operacijas vietoj įprastos matricų daugybos mažiausiai viename neuronų sluoksnyje.

Konvoliuciniai neuroniniai tinklai

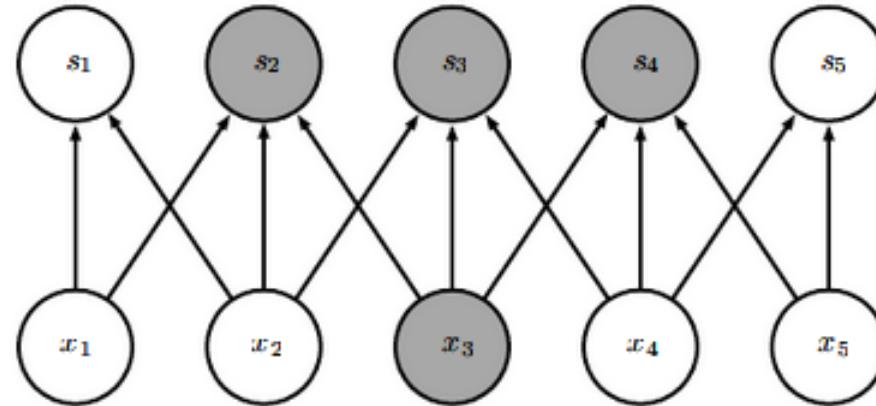
Šie konvoliucijos principai leidžia pagerinti mašininį mokymąsi:

- **išretinta sąveika,**
- **parametru pasidalinimas,**
- **ekvivalentiški atvaizdavimai** (reprezentacijos)

Konvoliuciniai sąryšiai tinkluose: išretinta sąveika



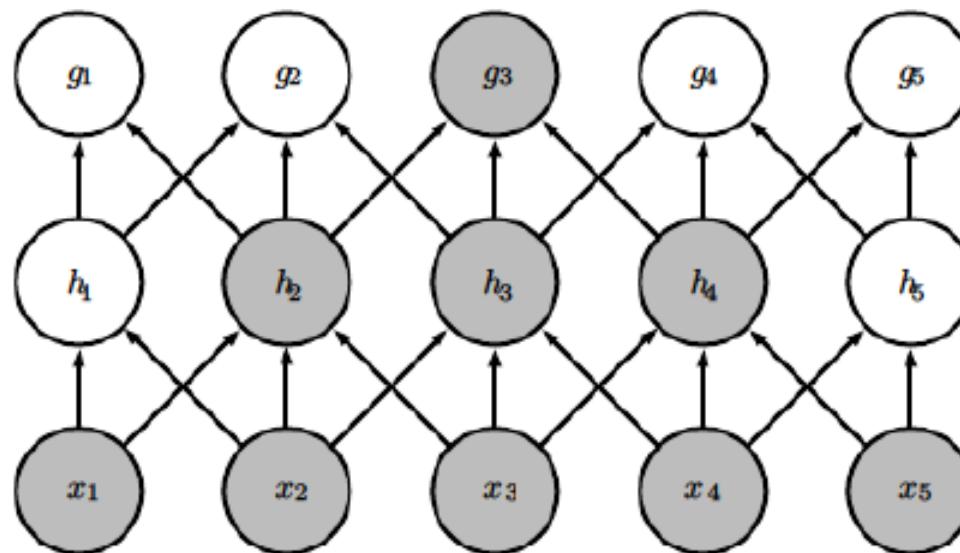
Tradiciniuose
tinkluose (**kiekvienas
su kiekvienu**)



Konvoliuciniuose
tinkluose (**kiekvienas
su tam tikrais**)

Konvoliuciniai sąryšiai tinkluose: išretinta sąveika

- Mazgai **gilesniuose sluoksniuose** gali netiesiogiai sąveikauti su didesne porcija įvesčių.
- Tai leidžia tinklui **efektyviau aprašyti sudėtingus sąryšius** tarp daug kintamųjų, sudarant tokias sąveikas iš paprastų blokų, kur kiekvienas aprašo tik išretintą sąveiką.



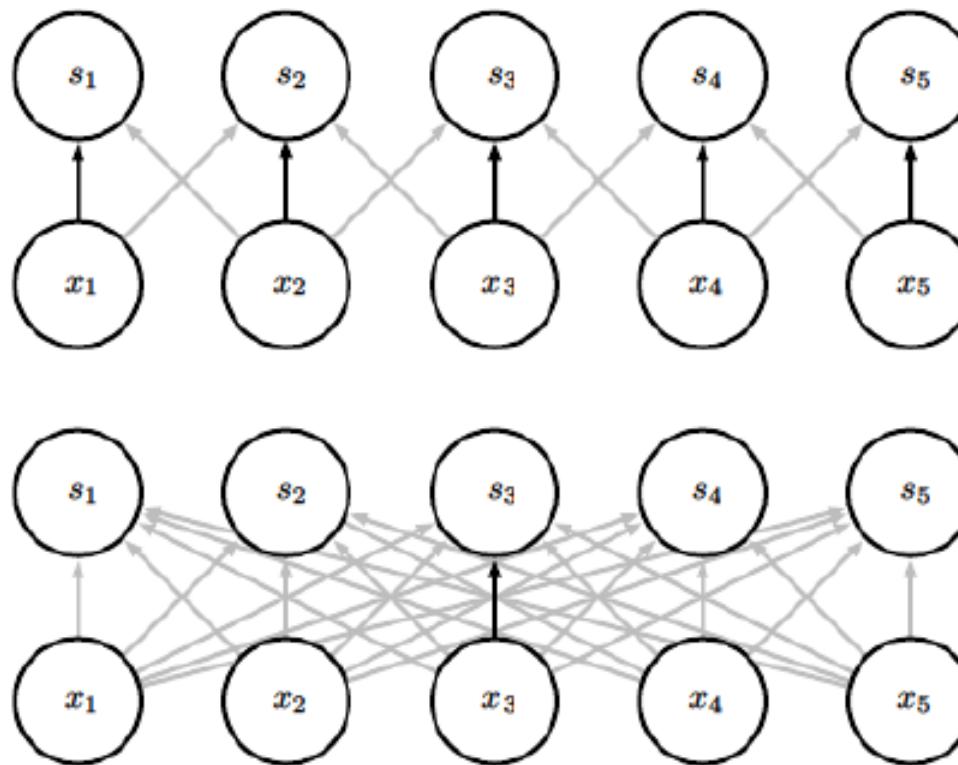
Vaizdų apdorojimas: išretinta sąveika

- Pavyzdžiui, įvesties vaizdas gali būti sudarytas iš tūkstančių ar net milijonų pikselių, tačiau galima nustatyti daug mažesnio skaičiaus, bet **reikšmingus požymius**, tokius kaip kraštai panaudojus branduoli, sudarytą tik iš dešimčių ar šimtų pikselių.
- Tai leidžia saugoti mažesnį kiekį informacijos, taip **sumažinamas būtinės atminties** kiekis, be to, padidinamas modelio statistinis patikimumas.

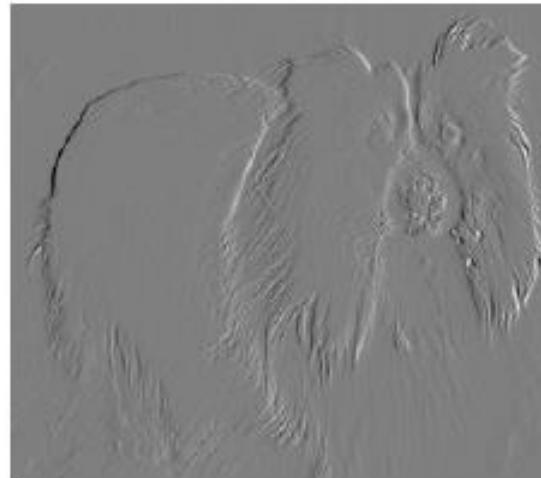
Parametru pasidalinimas

- Kai modelyje **tas pats parametras** naudojamas daugiau nei vienoje funkcijoje.
- **Tradiciniuose tinkluose** perskaičiuojant sluoksnio išėjimus, kiekvienas **svorių matricos** elementas yra **naudojamas tik vieną kartą**.
- **Konvoluciiniuose tinkluose** kiekvienas branduolio narys yra naudojamas kiekvienoje įvesties pozicijoje.
- Konvolucijoje naudojamas **parametru pasidalijimas** reiškia, kad vietoj to, kad kiekvienoje pozicijoje nagrinėti atskirus parametru rinkinius, nagrinėjamas vienas rinkinys.

Parametru pasidalinimas



Išretintos sąveikos ir parametru pasidalijimo įtaka kraštams vaizde nustatyti



Ekvivalentiškumas

- Sakoma, kad **funkcijos yra ekvivalentiškos**, kai pakeitus įvestį, tokiu pat būdu pasikeičia ir išvestis.
- Funkcija $f(x)$ yra **ekvivalenti** funkcija $g(x)$, jei $f(g(x)) = g(f(x))$.
- Konvoliucijos atveju parametru dalijimasis lemia tai, kad sluoksnis turėtų savybę, vadinamą **ekvivalentiškumu** pakeitimo operacijai.
- Tegu g yra bet kokia funkcija, perskaičiuojanti įeitį, pvz., patraukia ją, tuomet **konvoliucijos funkcija yra ekvivalenti** funkcija g .

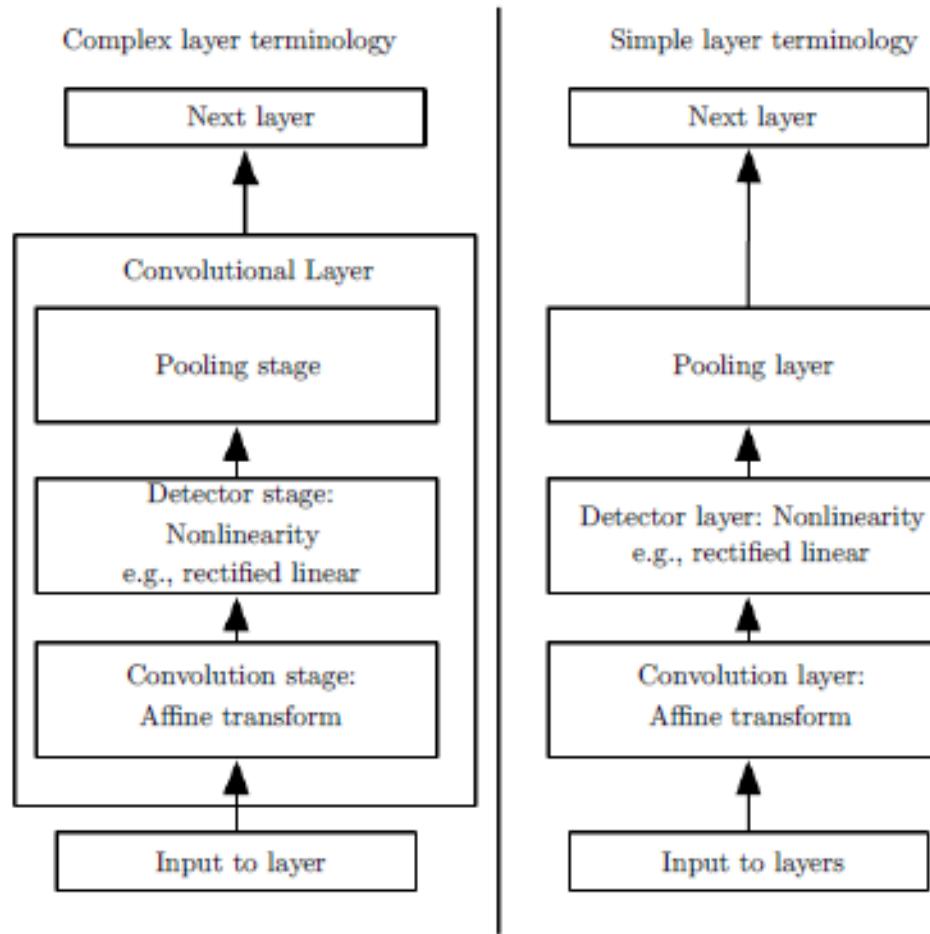
Ekvivalentiškumas

- Tegu I yra funkcija, apskaičiuojanti paveikslo ryškumą (angl. *brightness*), o g – funkcija, **atvaizduojanti vieną funkciją** į kitą taip, kad $I' = g(I)$ būtų funkcija, kur $I'(x, y) = I(x - 1, y)$, t.y. tokia funkcija pastumia kiekvieną pikselį per vieną poziciją į dešinę.
- Jei taikysime šią transformaciją funkcijai I , paskui taikysime konvoliucijos operaciją, **rezultatas bus toks pat**, kurį gautume pradžioje pritaikę konvoliuciją funkcijai I' , paskui transformaciją g .

Ekvivalentiškumas

- Tai naudinga tuomet, kai žinome, kad tam tikra nedidelio skaičiaus kaimyninių pikselių funkcija yra naudinga, kai taikoma **daugelyje įvesties vietų**.
- Pavyzdžiui, apdorojant vaizdus, naudinga **nustatyti briaunas** (kraštus) pirmame konvolucinio tinklo sluoksnuje.
- Briaunos yra beveik visur, taigi, tikslinga pasidalinti šia informacija **per visą paveikslėli**.
- Tačiau gali būti atveju, kai **nenorima dalintis parametrais** per visą vaizdą. Pavyzdžiui, apdorojant veido vaizdą, kiekviena jo dalis turi skirtingą informaciją.

Konvoliucinių neuroninių tinklų struktūra



Konvoliucinis sluoksnis

- Šio sluoksnio **pagrindinis tikslas** yra nustatyti požymių junginius iš ankstesnių sluoksninių ir atvaizduoti juos į **požymių žemėlapį** (angl. *feature map*).
- Vaizdas yra padalinamas, sukuriami lokalūs fragmentai ir vėliau jie sujungiami į **požymių žemėlapį**, kurio dydis $m_2 \times m_3$.
- Toks žemėlapis saugo informaciją, kaip gerai **požymis atitinka jam taikomą filtrą**.
- Kiekvienas filtras yra mokomas „erdviniu būdu“, atsižvelgiant į **trimatės struktūros vietą**, kuriai jis taikomas.

Konvoliucinis sluoksnis

- Kiekviename sluoksnyje yra m_1 **filtras**.
- Kiek filtrų bus taikoma priklauso nuo požymių žemėlapių **trimatės struktūros gylio**.
- Kiekvienas filtras aptinka **tam tikrą požymį**.
- l -ojo sluoksnio **išvestis** $Y_i^{(l)}$ yra sudaryta iš $m_1^{(l)}$ požymių žemėlapių, kurių dydis $m_2^{(l)} \times m_3^{(l)}$.

Konvoliucinis sluoksnis

- i -tasis požymių žemėlapis, žymimas $Y_i^{(l)}$,
apskaičiuojamas:

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)}$$

- Čia $B_i^{(l)}$ yra *bias* matrica
- $K_{i,j}^{(l)}$ yra **filtras** (branduolis), kurio dydis
 $2h_1^{(l)} + 1 \times 2h_2^{(l)} + 1$.
Jis sujungia j -tąjį požymių žemėlapį $(l - 1)$ -jame
sluoksnje su i -tuoj požymių žemėlapį l -jame
sluoksnje.

Konvoliucinis sluoksnis

- Šių konvoliucinių sluoksninių sujungimas su kitais sluoksniais **leidžia klasifikuoti informaciją** kaip tai atliekama **žmogaus regoje**.
- **Intuityviai suprantama**, kad iš pikselių sudaromi kraštai, iš kraštų formos, iš formų sudėtingesni objektai.

Netiesiškumo sluoksnis

- **Netiesiškumo sluoksnis** (angl. *non-linearity layer*) sudarytas iš aktyvacijos funkcijos, kuri paima požymiu žemėlapį, gautą konvoliucijos sluoksnyje, ir sukuria **aktyvacijos žemėlapį**.
- Kaip ir daugiasluoksniaiame perceptrone, dažniausios **aktyvacijos funkcijos**:
 - sigmoidinė
 - hiperbolinis tangentas
- Pastaruoju metu siūloma naudoti **ištaisymo tiesinę funkciją** (angl. *rectified linear unit* (ReLU)) (žr. tolesnes skaidres).

Netiesiškumo sluoksnis

- Tegu l yra **netiesiškumo sluoksnis**. Jis paima požymiu žemėlapį $Y_i^{(l-1)}$ ir generuoja aktyvacijos žemėlapį $Y_i^{(l)}$:

$$Y_i^{(l)} = f(Y_i^{(l-1)})$$

- $Y_i^{(l)} \in \mathbb{R}^{m_1^{(l)} \times m_2^{(l)} \times m_3^{(l)}}$,
- $Y_i^{(l-1)} \in \mathbb{R}^{m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}}$,
- $m_1^{(l)} = m_1^{(l-1)} \wedge m_2^{(l)} = m_2^{(l-1)} \wedge m_3^{(l)} = m_3^{(l)}$.
- Tai gali būti interpretuojama kaip $m_1^{(l-1)}$ **dvimačiai požymiu žemėliai**, kurie generuoti $m_1^{(l-1)}$ filtrų ($l - 1$)-ajame konvoliuciniame sluoksnyje. Šiu žemėlapiu dydžiai $m_2^{(l-1)} \times m_3^{(l-1)}$.

Ištaisymo sluoksnis

- Ištaisymo sluoksnis (angl. *rectification layer*) reikšmes pakeičia į **absoliučius** jų dydžius.
- Tarkime l yra ištaisymo sluoksnis. Jis paima aktyvacijos žemėlapį $Y_i^{(l-1)}$ iš $(l - 1)$ -ojo netiesiškumo sluoksnio ir sukuria **ištaisytaą aktyvacijos žemėlapį** $Y_i^{(l)}$:

$$Y_i^{(l)} = |Y_i^{(l-1)}|$$

Netiesiškumo ir ištaisymo sluoksnių apjungimas

Kaip ir netiesiškumo sluoksnyje, ištaisymo sluoksnyje požymių žemėlapių dydis nepakinta, todėl dažnai šie **sluoksniai apjungiami į vieną**:

$$Y_i^{(l)} = \left| f\left(Y_i^{(l-1)}\right) \right|$$

Ištaisymo tiesinė funkcija

- **Ištaisymo tiesinė funkcija** (angl. *rectified linear unit*, ReLU) yra specifinė funkcija, apjungianti netiesiškumo ir ištaisymo sluoksnius:

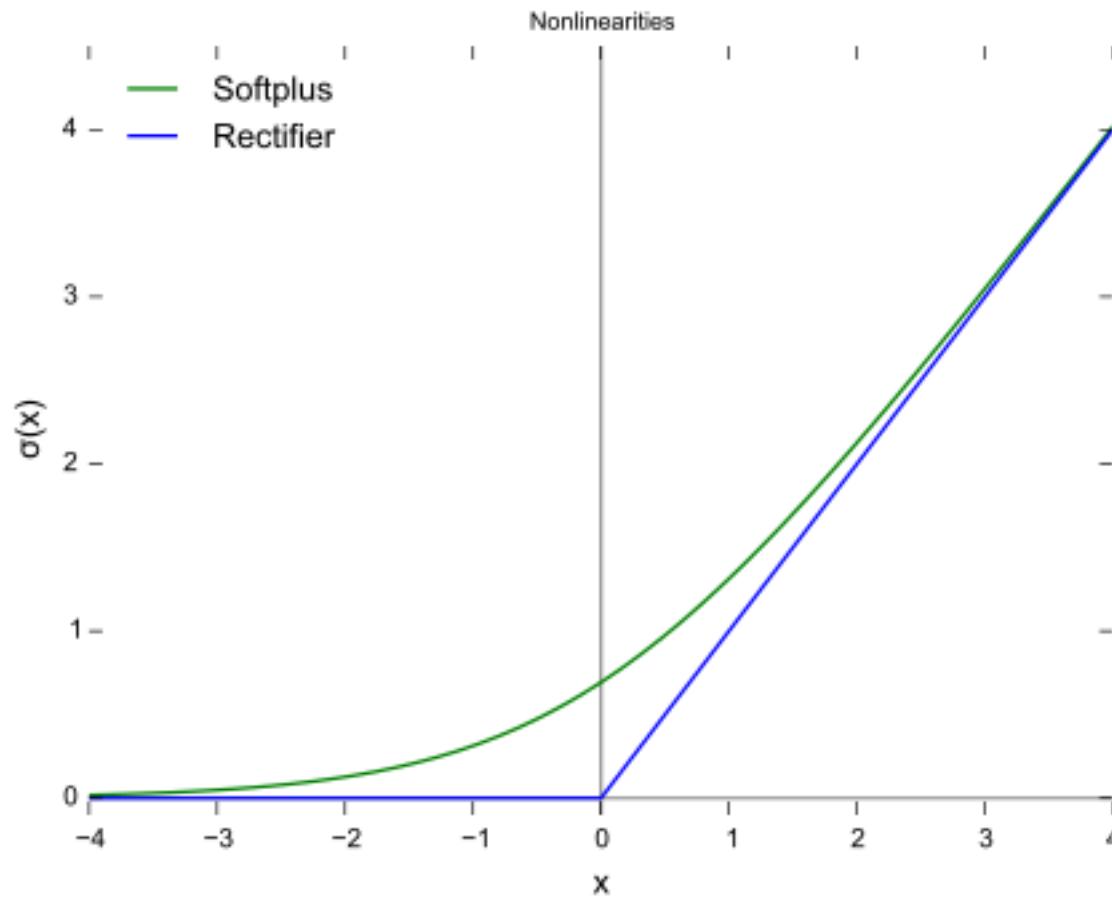
$$f(x) = \max(0, x).$$

- Galimi ir kiti variantai:
- Tolydi aproksimacija yra **softplus** funkcija:

$$f(x) = \ln(1 + e^x)$$

- Jos išvestinė $f'(x) = \frac{1}{1+e^{-x}}$ yra **sigmoidinė** (logistinė funkcija).

Softplus vs Rectifier



Ištaisymo tiesinė funkcija

Ištaisymo tiesinė funkcijos privalumai lyginant su sigmoidine ar hiperboliniu tangentu:

- Efektyviau skleidžiamas **gradientas**. Labai **paprastas gradiento** skaičiavimas (arba 0, arba 1, priklausomai nuo x ženklo).
- Dėl to kad neigiamos reikšmės prilyginamos nuliui, **išretinimas** tampa dar aktualesniu. Toks išretinimas yra naudingas siekiant atsparumo triukšmams.
- Operacijos labai **paprastos**, nėra **atliekama jokių sudėtingų veiksmų**.
- **Pagreitėja** neuroninio tinklo mokymas.

ReLU variantai

Dažniausiai taikomi ReLU variantai:

- **Su triukšmu** (angl. *noisy* ReLU):

$$f(x) = \max(0, x + Y), Y \sim \mathcal{N}(0, \sigma(x))$$

Sėkmingai naudojama apribotose Boltzmeno mašinose (angl. *restricted Boltzmann machines*) **kompiuterinės regos** uždaviniams spręsti

- **Nesandarus** (angl. *leaky* ReLUs):

$$f(x) = \begin{cases} x, & \text{jei } x > 0 \\ 0,01x, & \text{jei } x \leq 0 \end{cases}$$

čia leidžiami maži nenuliniai gradientai.

Sujungimo sluoksnis

- **Sujungimo sluoksnis** (angl. *pooling* arba *subsampling*) yra atsakingas už aktyvacijos žemėlapį dydžio sumažinimą.
- Jis taikomas po kelių konvoliucijos ir netiesiškumo sluoksnį siekiant **sumažinti skaičiavimų apimtis** ir minimizuojant persimokymą.

Sujungimo sluoksnis

- l -tasis **sujungimo sluoksnis** turi du parametrus: **filtras** $F^{(l)}$ ir **žingsnis** $S^{(l)}$.
- Jis paima trimatę įvestį, kurios dydis $m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}$ ir grąžina dydžio $m_1^{(l)} \times m_2^{(l)} \times m_3^{(l)}$ išvestį, kur:
 - $m_1^{(l)} = m_1^{(l-1)}$
 - $m_2^{(l)} = (m_2^{(l-1)} - F^{(l)})/S^{(l)} + 1$
 - $m_3^{(l)} = (m_3^{(l-1)} - F^{(l)})/S^{(l)} + 1$

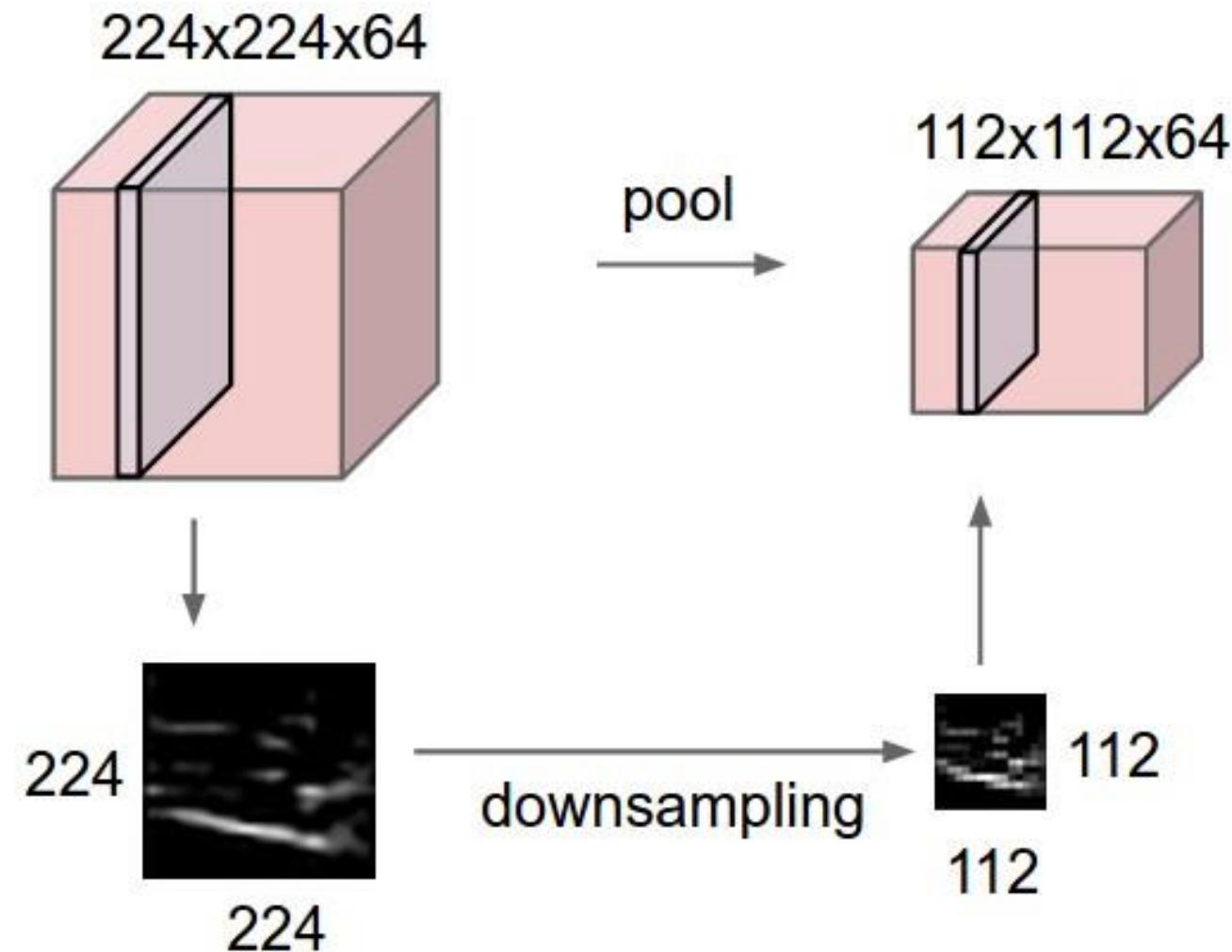
Sujungimo sluoksnis

- Pagrindinė idėja yra pateikti pakeitimo invariacią **atsižvelgiant į atpažinimo uždavinio** specifiką.
- **Požymiu nustatymas** yra daug svarbiau nei tiksliai požymiu vieta.
- Taigi, sujungimo sluoksnyje siekiama išsaugoti nustatytaus požymius, **atmetant mažiau svarbią** informaciją.

Sujungimo sluoksnis

- Apibrėžiamas langas $F^{(l)} \times F^{(l)}$ ir tokiu būdu **mažinimas duomenų kiekis** iki vieneto.
- **Langas yra traukiamas** per $S^{(l)}$ pozicijas. Duomenų mažinimas kartojamas kiekvienoje lango pozicijoje, kol sumažinamas įvesties erdvė.

Sujungimo sluoksnis

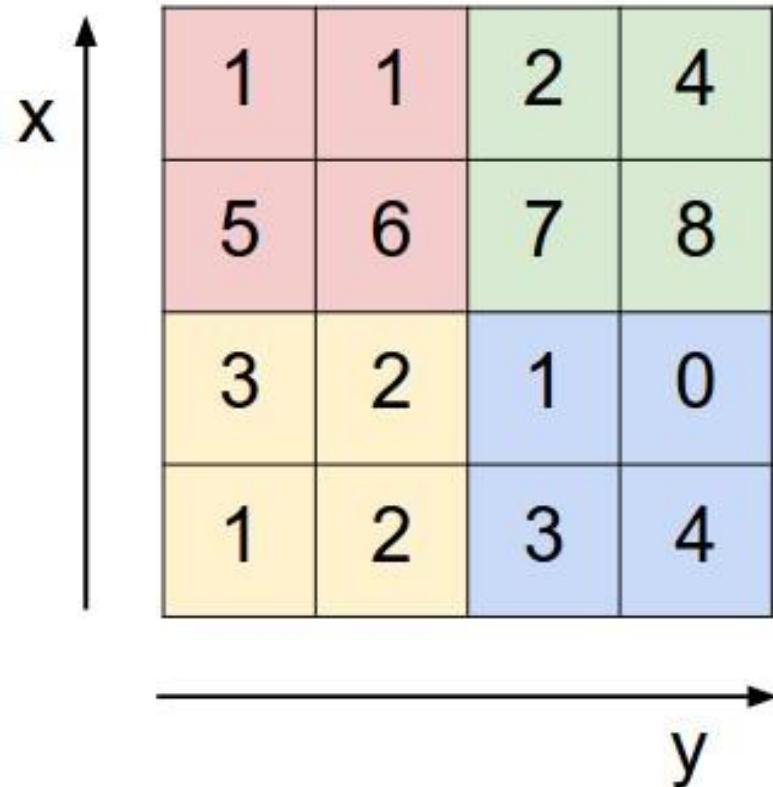


Sujungimo sluoksnis

- Dažniausi metodai: **maksimalus sujungimas** (angl. *max pooling*) ir **vidutinis sujungimas** (angl. *average pooling*).
- Maksimalaus sujungimo atveju ieškoma **didžiausios reikšmės** lange, vidutinio sujungimo – **vidutinės** reikšmės.
- Naudojant **maksimalų sujungimą**, tinklas greičiau apsimoko ir tiksliau sprendžia atpažinimo uždavinius.

Sujungimo sluoksnis

Single depth slice



max pool with 2x2 filters
and stride 2



6	8
3	4

Sujungimo sluoksnis

- Sujungimo sluoksnje be mažinimo metodo, svarbus ir **žingsnio** $S^{(l)}$ parinkimas. Jis nurodo, ar langas persidengs, ar ne.
- Jei $F^{(l)} > S^{(l)}$, **langai perdengs** vienas kitą, filtras bus taikomas kelis kartus toje pačioje vietoje.
- Priklausomai nuo šios sąlygos, sujungimo sluoksnis vadinamas **persidengiančiu** ar **nepersidengiančiu** (angl. *overlapping or non-overlapping pooling*)

Pilnai sujungtas sluoksnis

- **Pilnai sujungti sluoksniai** (angl. *fully connected layers*) praktiškai yra daugiasluoksniai perceptronai (dažniausiai dviejų ar trijų paslėptų sluoksnį), kurie atvaizduoja $m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}$ aktyvacijos trimates įvestis iš ankstesnių sluoksniu junginių **į klasį tikimybių pasiskirstymą**.
- Gaunama $m_1^{(l-i)}$ **išvesčių**, čia i nurodo perceptrono sluoksnį skaičių.

Pilnai sujungtas sluoksnis

- **Pagrindinis skirtumas** nuo standartinio daugiasluoksnio perceptrono yra tai, kad standartiniu atveju įvestis yra **vektorius**, konvoliucinių tinklų atveju – **trimatė struktūra**.

$$y_i^{(l)} = f(a_i^{(l)})$$

- Jei $(l - 1)$ yra pilnai sujungtas sluoksnis, tai:

$$a_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} w_{ij}^{(l)} y_j^{(l-1)}$$

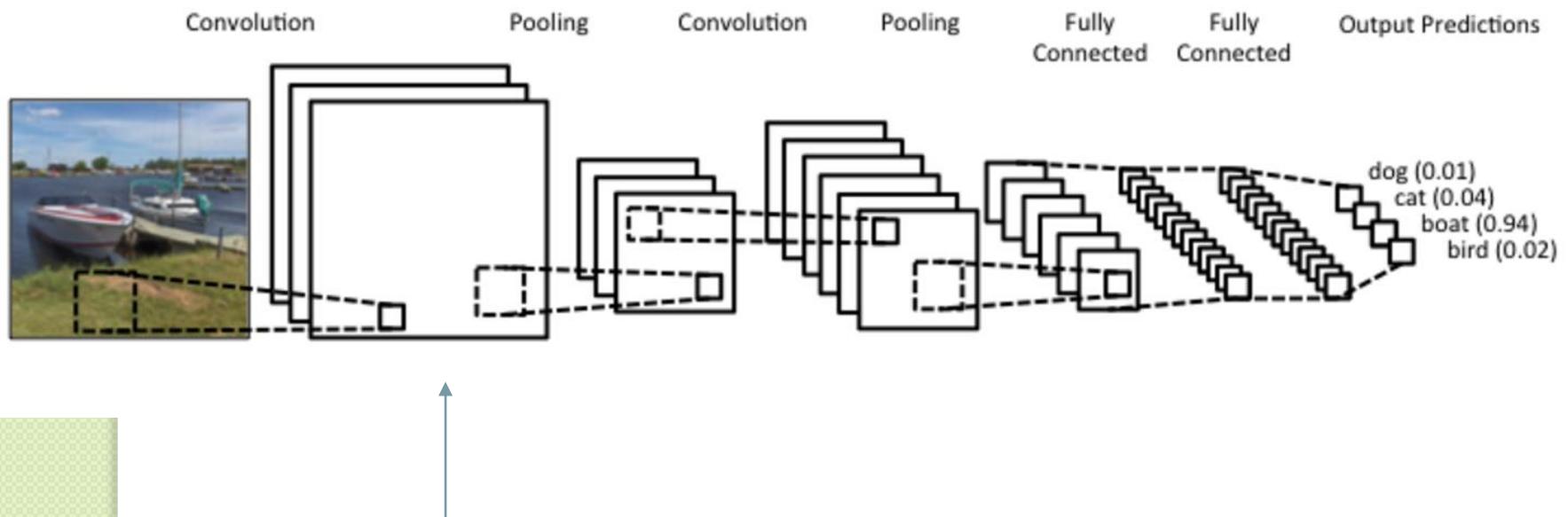
- Priešingu atveju:

$$a_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{ijrs}^{(l)} (Y_j^{(l-1)})_{rs}$$

Tinklo mokymas

- Belieka taip sudarytą tinklą **išmokyti**.
- Tai yra rasti tinkamas **svorių** $w_{ij}^{(l)}$ ir $w_{ijrs}^{(l)}$ reikšmes.
- $w_{ij}^{(l)}$ – svoriai pilnai sujungtame sluoksnyje (analogija su daugiasluoksniu perceptronu).
- $w_{ijrs}^{(l)}$ – filtrų reikšmės (ankstesnėse skaidrėse $K_{i,j}^{(l)}$)
- Siekiama nustatyti **kiekvienos klasės tikimybę**, pagrįstą aktyvavimo žemėlapiais, kurie sudaryti **konvolucijos, netiesiškumo, ištaisymo** ir **pilno sujungimo** sluoksnį.

Konvoliucinis neuroninis tinklas



Sluoksnių tiek, kiek
pritaikysime filtru

Konvoliucinis neuroninis tinklas

Paprastą ir lengvai suprantamą **apibendrinimą** galima rasti [https://github.com/brohrer/public-hosting/raw/master/How CNNs work.pdf](https://github.com/brohrer/public-hosting/raw/master/How%20CNNs%20work.pdf)

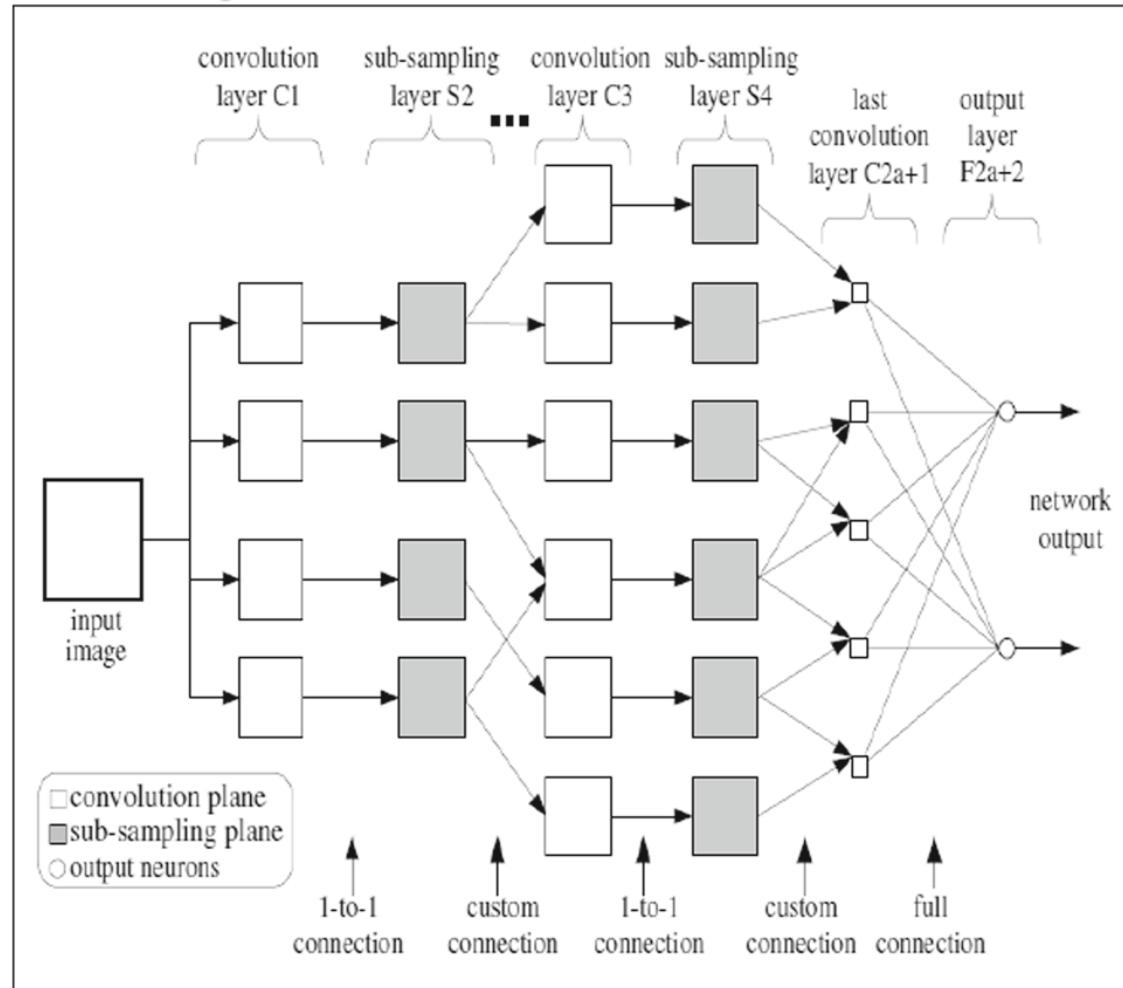
Konvoliucinio neuroninio tinklo sluoksniai

- **Konvolucijos** sluoksnis (**filtrai-svoriai**)
(*convolution layer*)
 - **Ištaisymo** sluoksnis (dažnai *ReLU*)
- **Sujungimo** sluoksnis (*pooling layer*,
downsampling layer, *sub-sampling layer*) (dažnai
max-pooling)
- P. S. Iprastai šie sluoksniai pasikartoja kelis kartus
- **Pilnai sujungtas** sluoksnis (*fully connected layer*,
dense layer)

Konvoliucinio sluoksnio parametrai

- **Filtrų skaičius**
- **Filtro** (*branduolio, kernel*) **dydis**
- **Lango žingsnis**: per kiek pikselių langas bus patrauktas (įprastai konvoliuciiniame sluoksnyje lygus 1, sujungimo (*pooling*) lygus 2)

Konvoliucinis neuroninis tinklas veidams atpažinti



Anuse, A., & Vyas, V. (2016). A novel training algorithm for convolutional neural network. *Complex & Intelligent Systems*, 2(3), 221-234

Svorų kiekis

- Tarkime įvestis – 74×74 paveiksliukas.
- C_1 – konvoliucinis sluoksnis su 6 filtrois,
kiekvienas jų yra 11×11 dydžio, dar yra 6 bias.
Svorų skaičius $11 \times 11 \times 6 + 6 = 732$.
- S_2 – sujungimo sluoksnis, neturintis svorų.
- C_3 – konvoliucinis sluoksnis su 16 filtro,
kiekvienas jų yra 11×11 dydžio, dar yra 16 bias.
Svorų skaičius $11 \times 11 \times 16 + 16 = 1952$.
- S_4 – sujungimo sluoksnis, neturintis svorų.

Svorių kiekis

- C_5 – konvoliucinis sluoksnis su 120 filtru, kiekvienas jų yra 11×11 dydžio, dar yra 120 bias. **Svorių skaičius** $11 \times 11 \times 120 + 120 = 14640$.
- F_6 – pilnai sujungtas sluoksnis, turintis 84 neuronus. **Svorių skaičius** $120 \times 84 = 10080$.
- Tinklas skirtas 25 asmenims atpažinti, tad išvesties sluoksnis sudarytas iš 25 neuronų. **Svorių skaičius** $85 \times 25 = 2100$.
- Viso svorių **29504**.

Konvoliucinio neuroninio tinklo mokymas

- Įprastai **tiesioginio sklidimo neuroniniai tinklai** mokomi minimizuojant paklaidos funkciją, taikant **gradientiniu nusileidimu** grįstus algoritmus, pavyzdžiui, klaidos skleidimo atgal.
- **Konvoliuciniai neuroniniai tinklai** mokomi pagal **stochastiniu gradientiniu nusileidimu** grįstus algoritmus. Čia taip pat gali būti taikoma klaidos skleidimo atgal algoritmo strategija.

Gradientinis nusileidimas

- Tarkime norime minimizuoti paklaidą

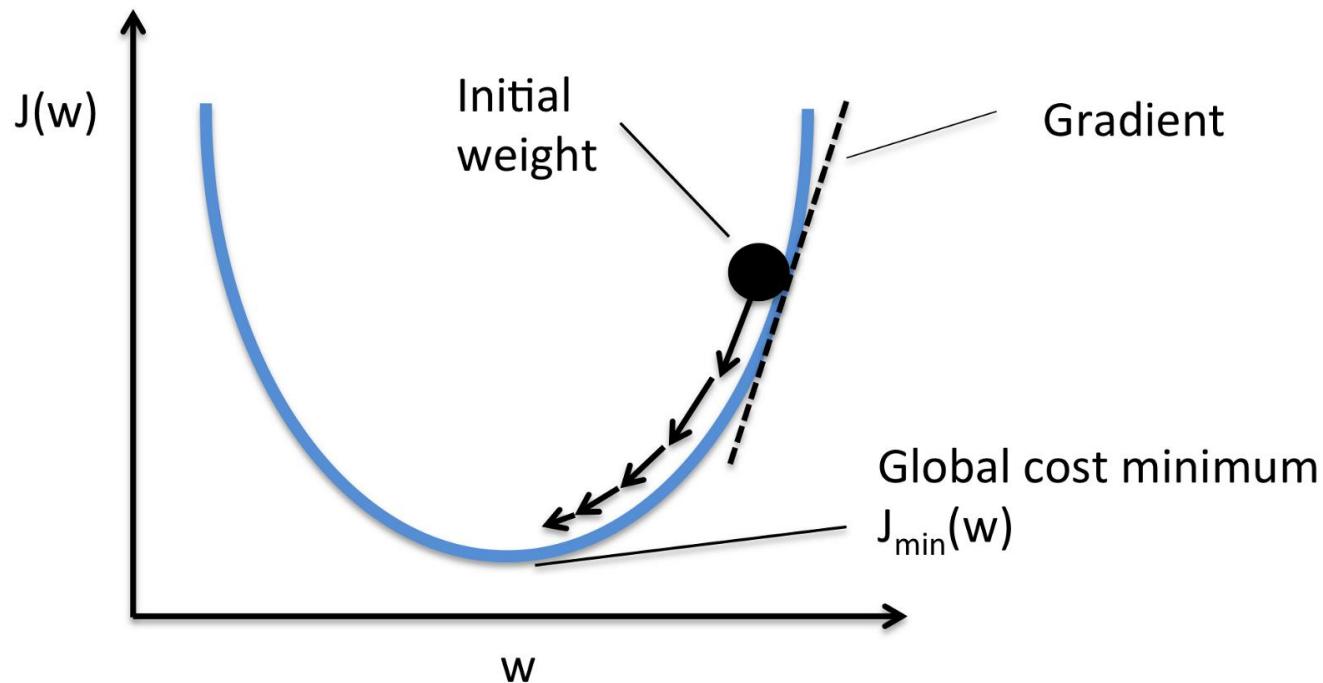
$$E(W) = \frac{1}{m} \sum_{i=1}^m E_i(W)$$

- Taikant **gradientinio nusileidimo algoritma**, svoriai keičiami pagal formulę:

$$\begin{aligned} W(t+1) &= W(t) - \eta \nabla E(W) = \\ &= W(t) - \eta \frac{1}{m} \sum_{i=1}^m \nabla E_i(W). \end{aligned}$$

- T. y. siekiama, kad paklaida būtų **minimali visiems mokymo duomenims**.

Gradientinis nusileidimas



Judama antigradiento kryptimi.

Stochastinis gradientinis nusileidimas

- **Stochastinio gradientinio nusileidimo** (*stochastic gradient descent*, SGD) algoritme tikras funkcijos $E(W)$ gradientas aproksimuojamas gradientu, gautu pagal vieną mokymo duomenų įrašą:

$$W(t + 1) = W(t) - \eta \nabla E_i(W)$$

- T. y. siekiama, kad paklaida būtų **minimali *i*-tajam mokymo duomenų** įrašui.

Stochastinis gradientinis nusileidimas + *momentum*

- Jei **stochastinio gradientinio nusileidimo** algoritme taikomas ***momentum*** metodas, svoriai keičiami pagal formule:

$$W(t + 1) = W(t) - \eta \nabla E_i(W) + \alpha \Delta W$$

- Čia ΔW – svorių pokytis gretimose iteracijose.

Paketo sąvoka neuroniniuose tinkluose

- Neuroniniuose tinkluose dažnai sutinkama **paketo (*batch*)** sąvoka, o tiksliau jos dydis (*batch size*).
- Iprastai apibrėžiama, kad **paketo dydis** – tai objektų, pateikiamų į tinklą, skaičius vienos iteracijos metu. Be to, tai parodo, keliems objektams reikia **verti gradientą** keičiant svorius.
- Taikant tradicinį **SGD** optimizavimo algoritmą, paketo dydis lygus **1**.
- Tačiau gali būti naudojamas ir modifikuotas SGD, kai gradientas vertinamas ne visiems mokymo duomenimis (kaip yra gradientinio nusileidimo algoritme), o tik **esantiems einamajame pakete**.

Adam algoritmas

- Dažnai nuo tinklo mokymo (**optimizavimo**) algoritmo priklauso **mokymo trukmė** (kuri gali svyruoti nuo kelių minučių iki kelių dienų).
- **Adam algoritmas** yra vienas populiariausių algoritmų, naudojamų giliesiems neuroniniams tinklams mokyti.
- Tai **praplėstas stochastinio** gradientinio nusileidimo algoritmas.
- „the name **Adam** is derived from adaptive moment estimation.“

Kingma, D. P., & Ba, J. (2015). Adam: a method for stochastic optimization. In Conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015, <https://arxiv.org/abs/1412.6980>

Adam algoritmas

- Adam algoritmas apjungia dviejų metodų privalumus: AdaGrad (*Adaptive Gradient Algorithm*) ir RMSProp (Root Mean Square Propagation)
- Iprastai stochastinio gradientinio nusileidimo metode naudojama vienintelė mokymo greičio parametro reikšmė (*learning rate*) visų svorių keitime ir ji išlieka pastovi viso mokymo metu.
- Adam algoritme mokymo greičio parametras derinimas mokymo procese.

Algorithm 1: Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

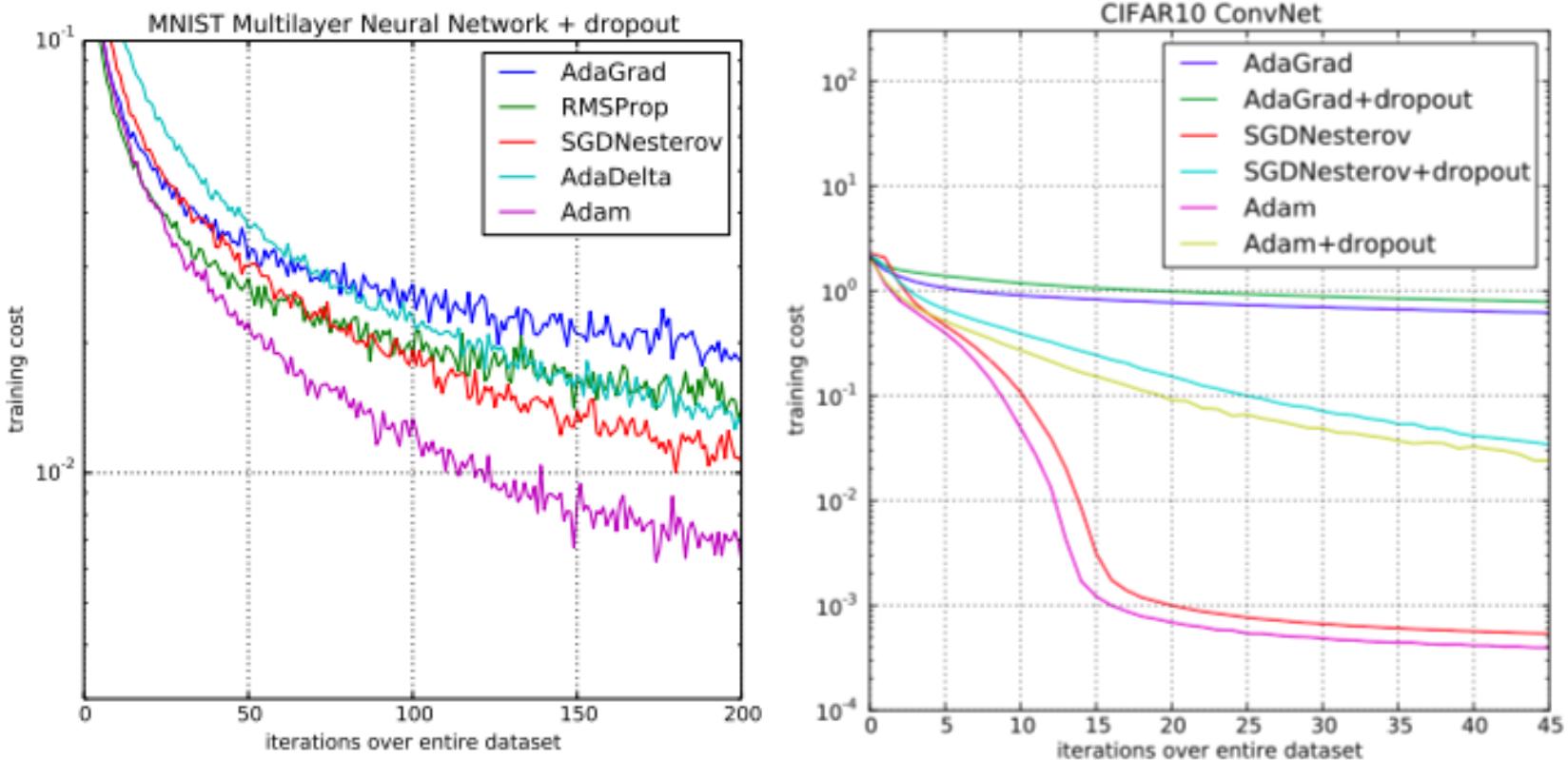
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t (Resulting parameters)

Kingma, D. P., & Ba, J. (2015). Adam: a method for stochastic optimization. In Conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015, <https://arxiv.org/abs/1412.6980>

Adam algoritmas



Kingma, D. P., & Ba, J. (2015). Adam: a method for stochastic optimization. In Conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015, <https://arxiv.org/abs/1412.6980>

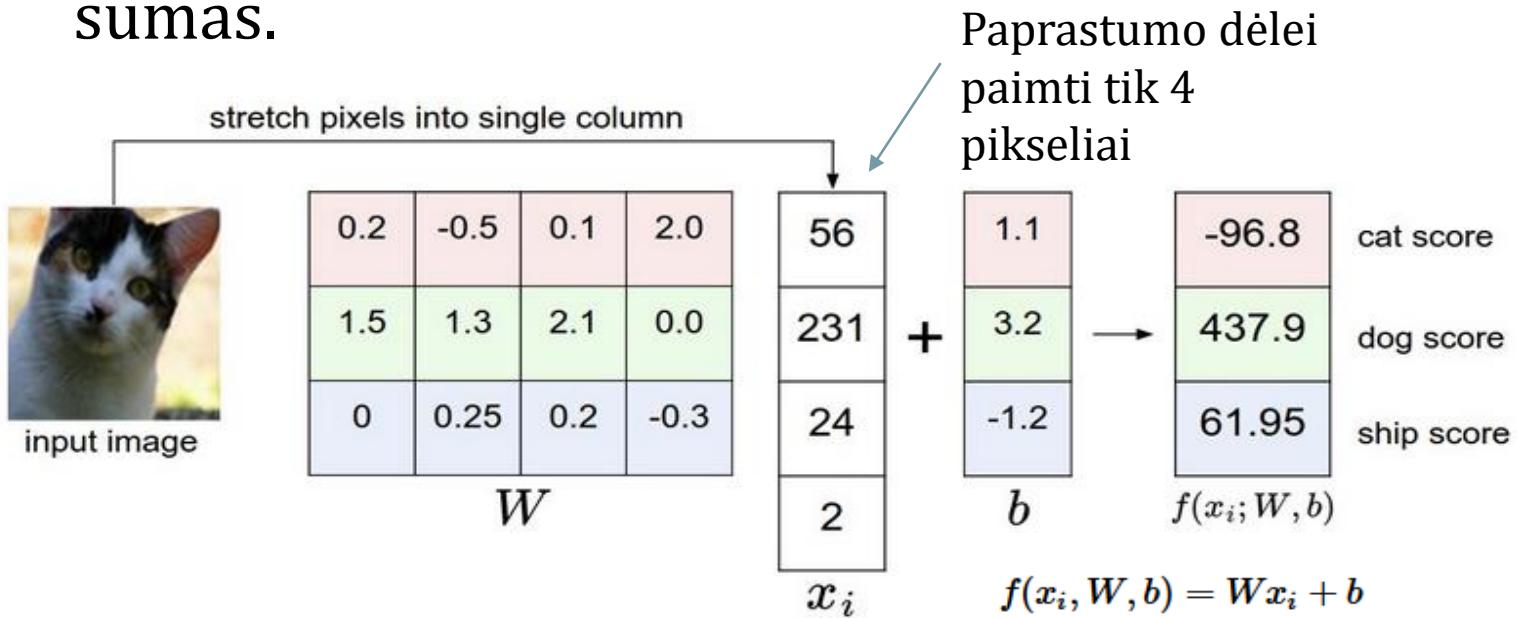
Optimizavimas neuroniniuose tinkluose (apibendrinimas)

- Optimizavimo metu tinklas **mokamas** (parenkami tinkami svoriai)
- **Algoritmai:**
 - Gradientinis nusileidimas (be/su *momentum*)
 - Stochastinis gradientinis nusileidimas (SGD) (be/su *momentum*) (su *Nesterov momentum*)
 - AdaGrad algoritmas
 - RMSProp algoritmas
 - Adam algoritmas
 - kiti

Daugiau info: <http://ruder.io/optimizing-gradient-descent/>

Duomenų priskyrimas klasėms

- **Tiesinis klasifikatorius** skaičiuoja kiekvienos klasės balus – visų pikselių reikšmių svertinės sumas.



Čia svoriai nėra tinkami, kadangi pagal išėjimų reikšmes nustatoma (didžiausia reikšmė 437,9), kad tai šuo (o ne katinas).

Iš <http://cs231n.github.io/linear-classify/>

Vaizdų pirminis apdorojimas

- Naudojant **RGB** spalvų sistemą, kiekvienas vaizdo pikselis gali įgauti reikšmes intervale **[0 ... 255]**.
- Dažnai reikia vaizdus **normuoti**.
- Vienas įprastų būdų – „**centravimas**“, kurio metu iš kiekvienos pikselio reikšmės atimamas vidurkis. Tuomet reikšmės atsiduria apytiksliai intervale **[-127 ... 127]**.
- Tolesnė procedūra pakeičia reikšmes, taip kad būtų intervale **[-1 ... 1]**.

Nuostolių funkcija

- **Nuostolių funkcija** (angl. *loss function, cost function, objective function*) reikalinga siekiant pamatuoti, kaip blogai (gerai) klasifikuojami duomenys.
- Kuo šios funkcijos **reikšmė didesnė**, tuo **klasifikavimas prastesnis**.

Daugiaklasių SVM nuostolių funkcija

- **Daugiaklasių atraminių vektorių klasifikatoriaus nuostolių funkcija** (angl. *Multiclass Support Vector Machine (SVM) loss*) sudaryta taip, kad SVM siekia, kad **teisingos klasės balas būtų aukštesnis** nei neteisingos, taikant tam tikrą fiksotą dydį Δ .
- Tarkime, i -tajam duomenų įrašui turime vaizdą x_i ir klasės žymę y_i .
- Funkcija $f(x_i, W)$ apskaičiuoja klasės balus kiekvienai klasei $s_j = f(x_i, W)_j, j = 1, \dots, K$. Čia K – klasių skaičius.
- Daugiaklasių SVM **nuostolių funkcija (loss)** i -tajam duomenų įrašui (ji turi būti minimizuojama)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Pavyzdys

- Tarkime turime **tris klasses** ir jų balai $s = [13, -7, 11]$. Teisinga klasė yra **pirmoji**. Tarkime $\Delta = 10$.
- **Gauname** $L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10) = 0 + 8$
- Taigi, siekiama, kad teisingos klasės balas būtų didesnis nei neteisingos **dydžiu** Δ .



Reguliavimas

- Aptarta nuostolių funkcija turi **didelį trūkumą**. Tarkime, kad visiems mokymo duomenų įrašams nuostolių paklaida lygi nuliui. Vadinasi, svorių reikšmių aibė W **nėra vienintelė**.
- Jei esant svoriams W , paklaidos nulinės, tai jos **išliks nulinės ir esant svoriams λW , $\lambda > 1$** . Tokiu būdu svorių reikšmės gali labai išdidėti.
- Tam įvedamas **reguliavimo bauda** (angl. *regularization penalty*).

Reguliavimas

- Dažniausiai naudojama **L1 arba L2 norma:**

$$\text{L1: } R(W) = \sum_k \sum_l |W_{kl}|.$$

$$\text{L2: } R(W) = \sum_k \sum_l W_{kl}^2.$$

- Tuomet **nuostolių funkcija:**

$$L = \frac{1}{m} \sum_i L_i + \lambda R(W).$$

Δ reikšmės nustatymas

- Ši reikšmė gali būti parenkama **kryžminės patikros** metu.
- Dažnai **keičiamos reikšmės** nuo $\Delta = 1$ iki $\Delta = 100$.

Softmax funkcija

- Konvoluciiniuose neuroniniuose tinkluose klasių priskyrimui dažnai naudojama funkcija yra **softmax funkcija**, kuri yra logistinės regresijos apibendrinimas daugeliui klasių:

$$f_i(Z) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}, i = 1, \dots, K$$

- Čia K – klasių skaičius.
- Ši funkcija **mažesnes** reikšmes **sumažina, didesnes padidina**.
- Pvz., $[1 \ 2 \ 3 \ 4 \ 1 \ 2 \ 3] \rightarrow$
 $\rightarrow [0,024 \ 0,064 \ 0,175 \ 0,475 \ 0,024 \ 0,064 \ 0,175]$

Kryžminė entropija

- Taikant *softmax* funkciją, naudojama **kryžminės entropijos** (angl. *cross-entropy*) nuostolių funkcija kiekvienam i -tajam duomenų įrašui (ji turi būti minimizuojama):

$$L_i = -\log \left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right)$$

- **P.S. Entropija** yra informacijos teorijoje naudojamas dydis, kuris apibūdina vidutinį informacijos kieki, kurį teikia vienas pranešimas (*Wikipedia*).

Kryžminė entropija

- Informacijos teorijos požiūriu, **kryžminė entropija** tarp tikrojo pasiskirstymo p ir i gauto pasiskirstymo q yra apibrėžiama taip:

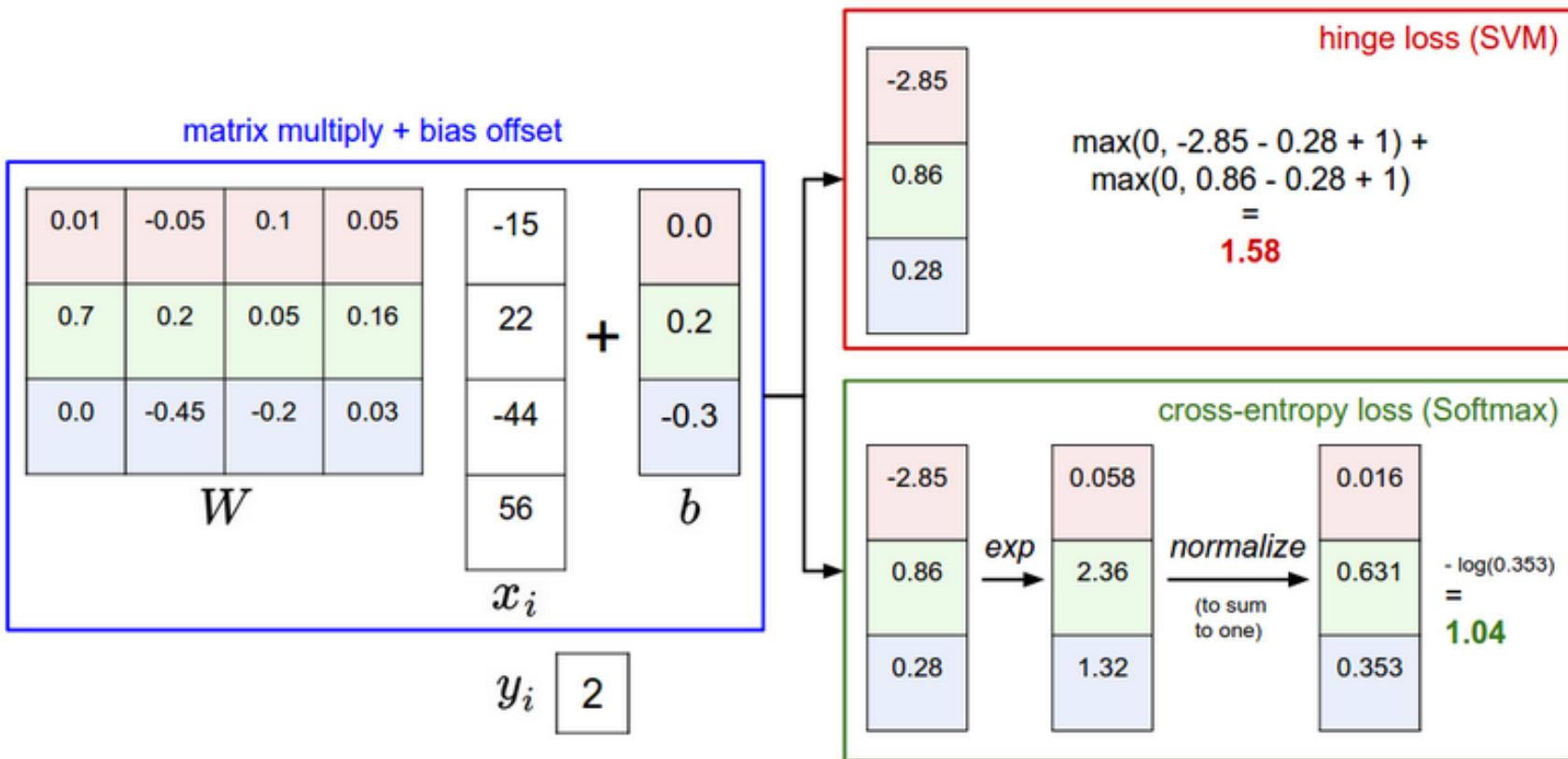
$$H(p, q) = - \sum_x p(x) \log q(x).$$

- Taikant **softmax** klasifikatoriu, minimizuojant **kryžminę entropiją**,
gauta klasių tikimybė $q = \frac{e^{sy_i}}{\sum_j e^{sj}}$,
tikrasis klasių pasiskirstymo $p = [0, \dots 1, \dots 0]$, kurį sudaro nuliukai ir vienas vienetas y_i pozicijoje.

Nesusipainiokime ...

- SVM klasifikatoruje – hinge loss, max-margin loss
- Softmax klasifikatoruje – cross-entropy loss

SVM vs. Softmax



Tinklo persimokymas

- Tinklo **persimokymas** (angl. *overfitting*) dažnai įvyksta, kai duomenų imtis salyginai maža, palyginus su parametru, kuriuos reikia nustatyti mokymo metu (išmokyti), skaičiumi.
- Ši problema ypač aktuali **giliojo mokymosi neuroniniuose tinkluose**.

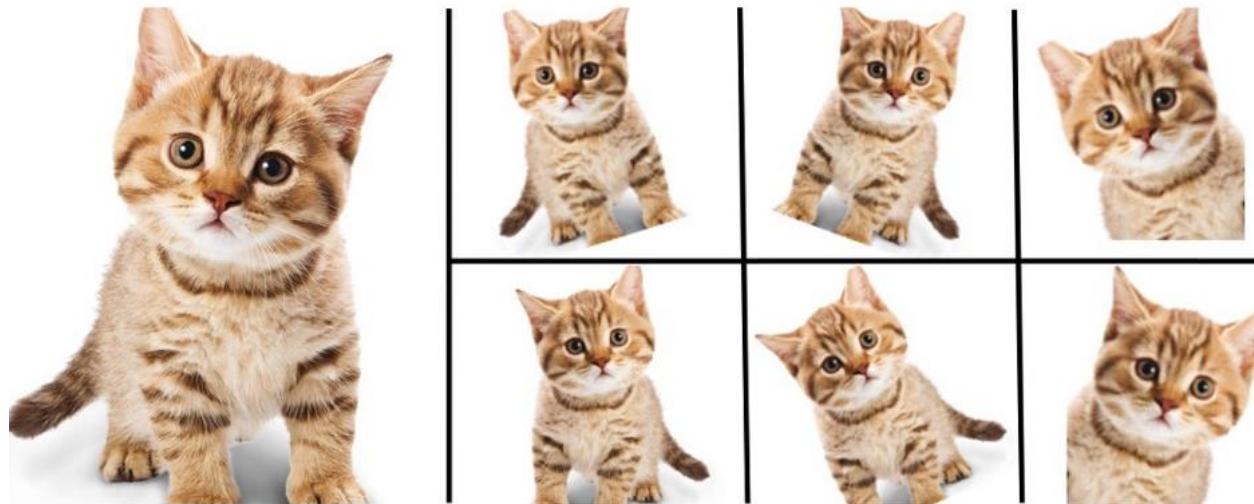
Tinklo persimokymas

Siekiant išvengti tinklo permokymo, taikomos **kelios strategijos**:

- Duomenų padidinimas (*augmentation*),
- Ankstyvas mokymo sustabdymas,
- **Išmetimo sluoksnis** (*dropout*),
- Svorų baudos L1 ir L2.

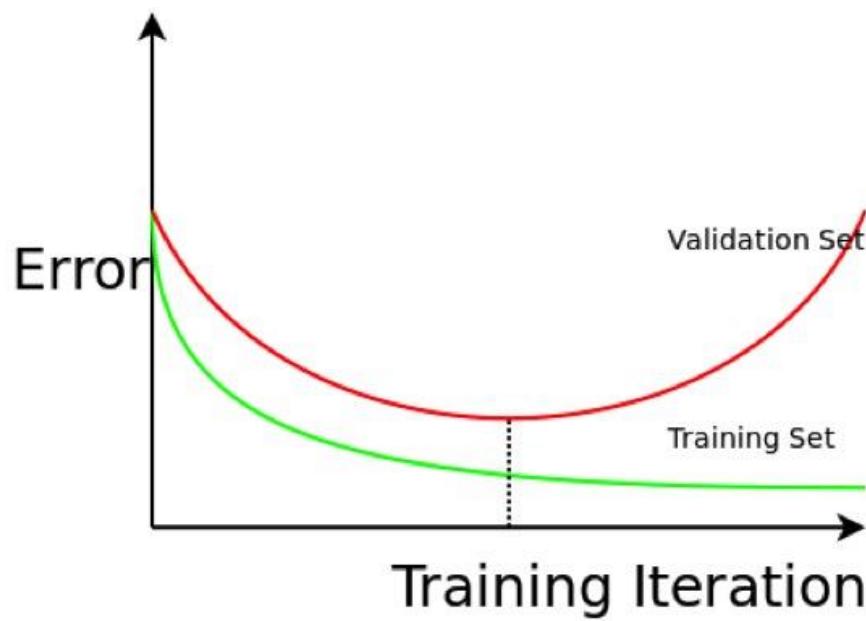
Tinklo persimokymas: duomenų padidinimas (*augmentation*)

- Siekiant gautu daugiau mokymo duomenų, iš turimų yra **sintetinami nauji**.
- Gali būti **taikomi įvairūs būdai**, pavyzdžiui, paveiksliuke atliekamos kelių pikselių transformacijos, tokios kaip pasukimas, mastelio keitimas ir kt.



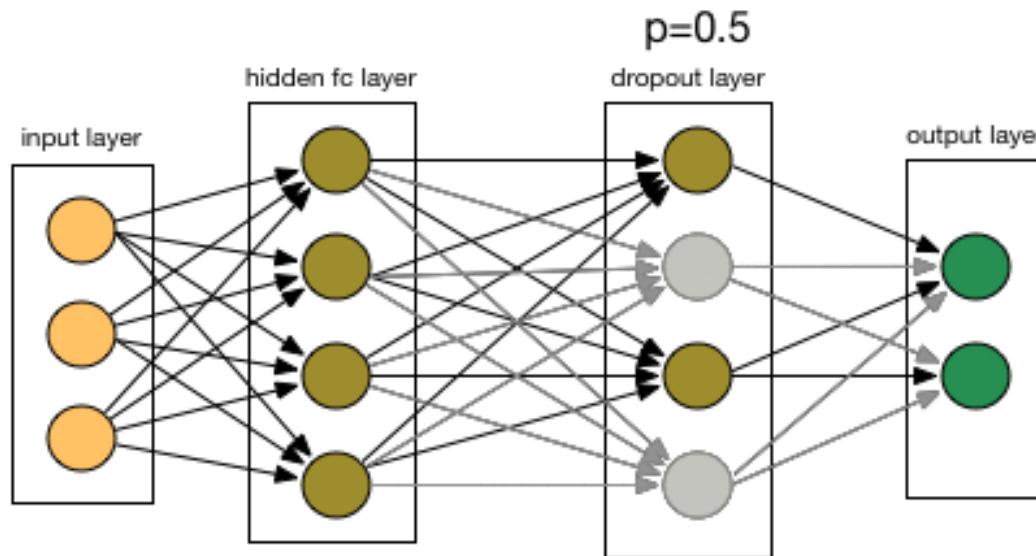
Tinklo persimokymas: duomenų padidinimas, ankstyvas stabdymas

- Tinklo mokymą **reikia stabdyti** tuomet, kai nors paklaida mokymo duomenims mažėja, bet jau pradeda **didėti paklaida validavimo** (testavimo) duomenims.



Tinklo persimokymas: išmetimo sluoksnio naudojimas

- Kiekvienoje mokymo iteracijoje **išmetimo** (angl. *dropout*) sluoksnis atsitiktinai **išmeta** kelis neuronus.



Tinklo persimokymas: išmetimo sluoksnio naudojimas

Kodėl išmetimo **sluoksnis veiksmingas**?

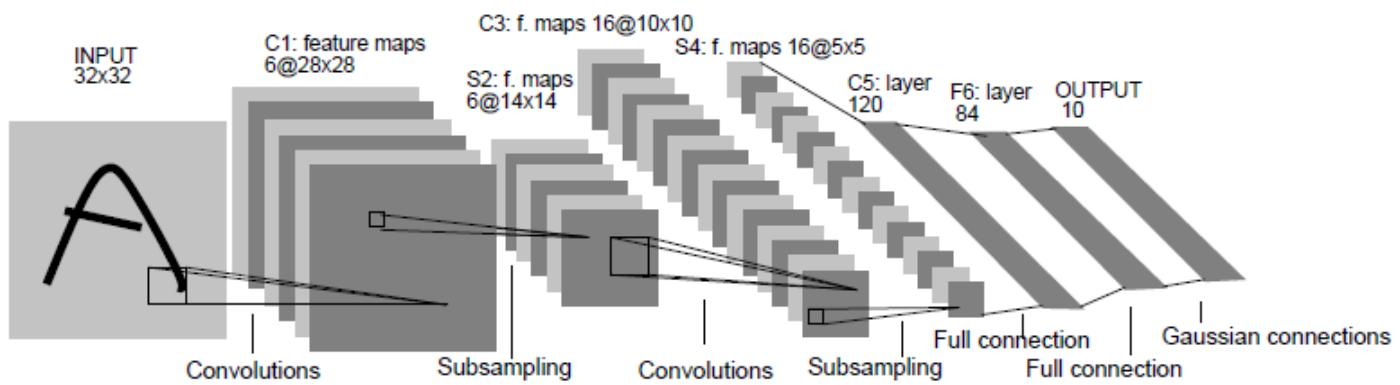
- Neuronai tampa **mažiau jautrūs** kitų neuronų svoriams, taigi, modelis tampa *robastiškas*.
- Išmetimo sluoksnis gali atitikti kelių modelių apibendrinimą, vadinama **ansambliu** (angl. *ensemble*). Pastaruoju metu ansambliai gauna tiksliausius mašininio mokymosi rezultatus.

Apibendrinimas

- Konvoliucijos sluoksniai (**filtrai-svoriai**)
- Ištaisymo sluoksnis (rectified linear units (ReLUs))
- Maksimalaus **sujungimo** sluoksnis (max-pooling)
- **Pilnai sujungtas** sluoksnis (fully connected layer)
- **Soft-max** klasifikavimas

Sėkminges tinklai: LeNet

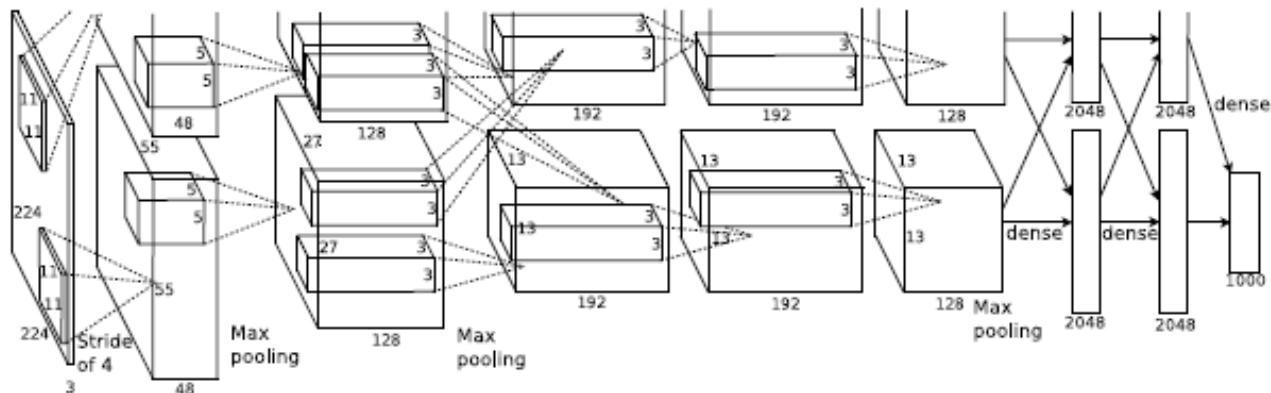
- **LeNet** – autoriai Yann LeCun ir kt., 1998,
rašytiems skaitmenims atpažinti



LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

Sėkminges tinklai: AlexNet

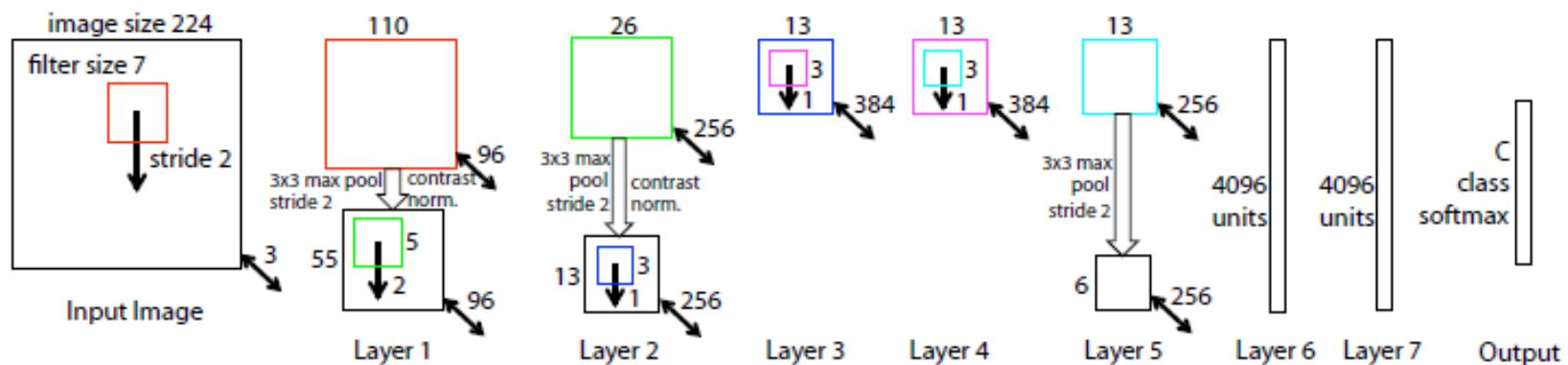
- **AlexNet** – autorai Alex Krizhevsky, Ilya Sutskever ir Geoff Hinton.
- Tai pirmasis darbas, išpopuliarinęs CNN. Dalyvavo konkurse „ImageNet **ILSVRC Challenge** in 2012“



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Sékmingsi tinklai: ZF Net

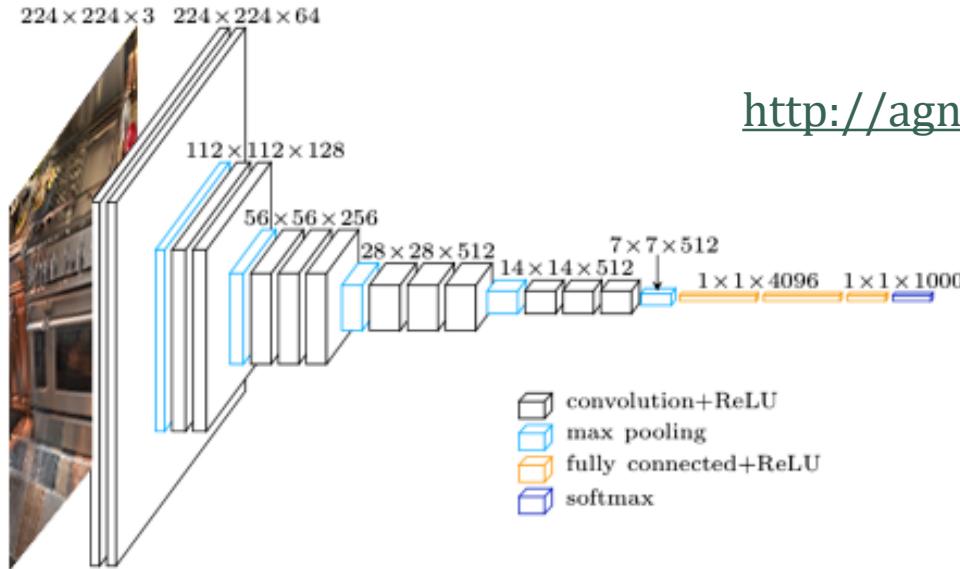
- **ZF Net** – autoriai Matthew Zeiler ir Rob Fergus.
- ILSVRC 2013 (Large Scale Visual Recognition Challenge) nugalėtojas.



Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

Sėkminges tinklai: VGGNet

- **VGGNet** – autoriai Simonyan, K. ir Zisserman ILSVRC 2014 antrosios vienos nugalėtojas.



Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. <https://arxiv.org/abs/1409.1556>

Sėkminges tinklai: ResNet

- **ResNet** (Residual Network) – autorai Kaiming He et al. ILSVRC 2015 nugalėtojas.

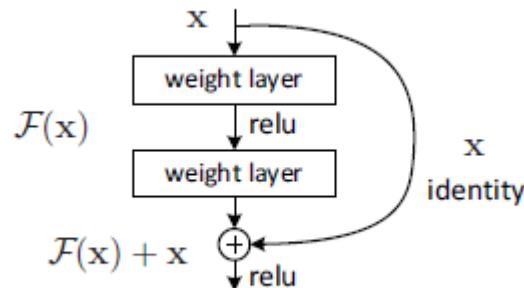
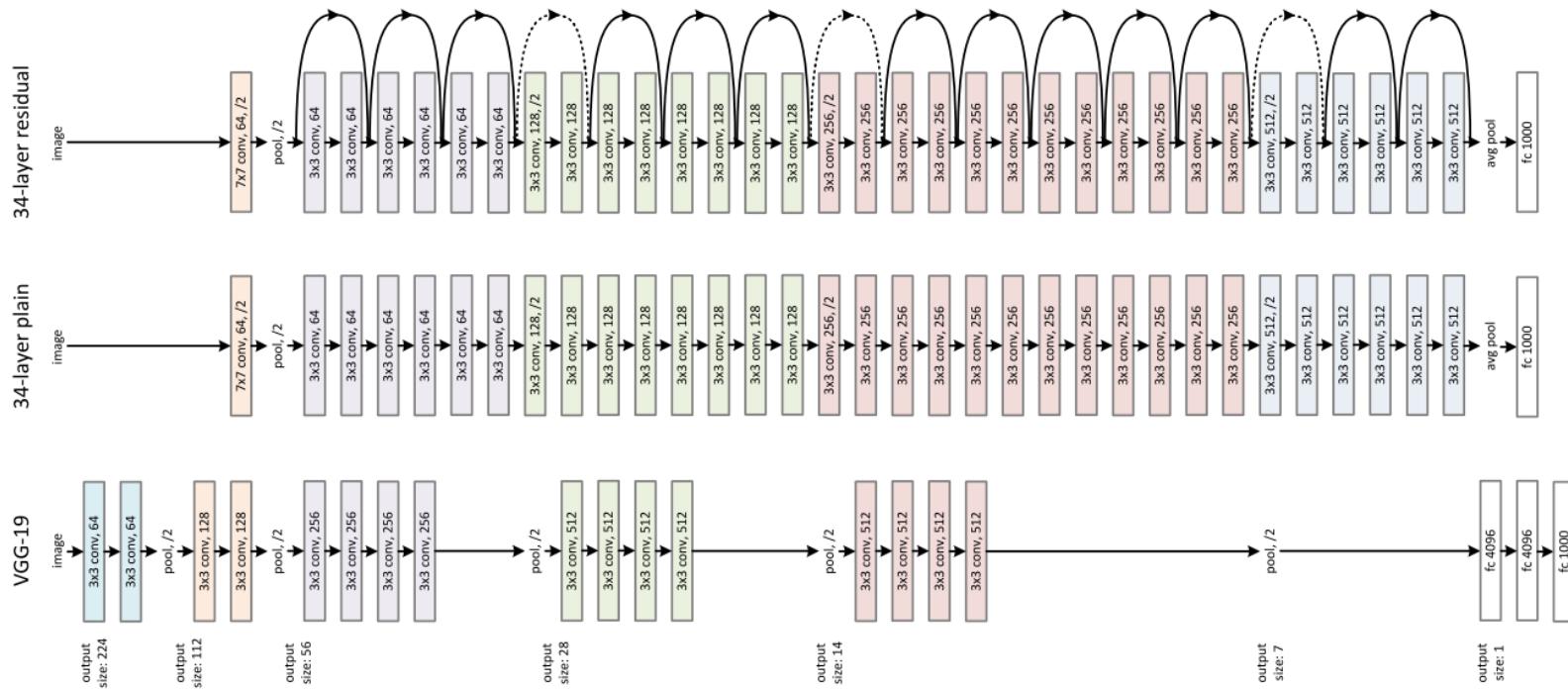


Figure 2. Residual learning: a building block.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
<https://arxiv.org/abs/1512.03385>

Sékmingsi tinklai: ResNet

- **ResNet** (Residual Network) – autoriai Kaiming He et al. ILSVRC 2015 nugalėtojas.



He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
<https://arxiv.org/abs/1512.03385>

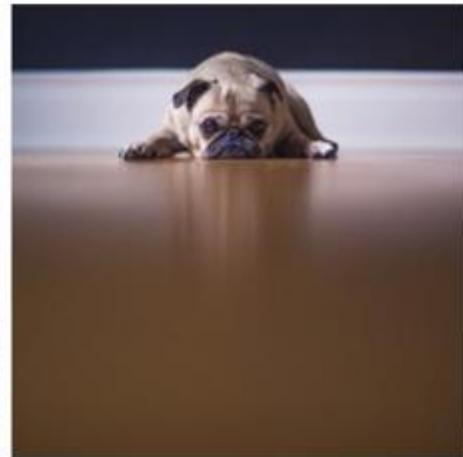
Sėkmingi tinklai: GoogleNet

- **GoogleNet/InceptionV1** – konkurso ILSVRC2014 nugalėtojas. Dabar jau yra **Inception V4** versija.
- Vienas iš būdų **pagerinti** giliųjų neuroninių tinklų veikimą – **padidinti** jų dydį.
- Tačiau tuomet reikia ir **didinti** mokymo duomenų aibę. Be to, stipriai **padidėja** skaičiavimo apimtys.
- **Google** komanda pasiūlė būdą pereiti iš pilnai jungaus į išretinto jungimo architektūrą.

Szegedy C., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Sėkmingi tinklai: GoogleNet

- **GoogleNet / Inception** sluoksnio idėja – padengti didesnį plotą, tačiau išlaikant vaizduose **gerą rezoliuciją** mažam kiekiui informacijos.

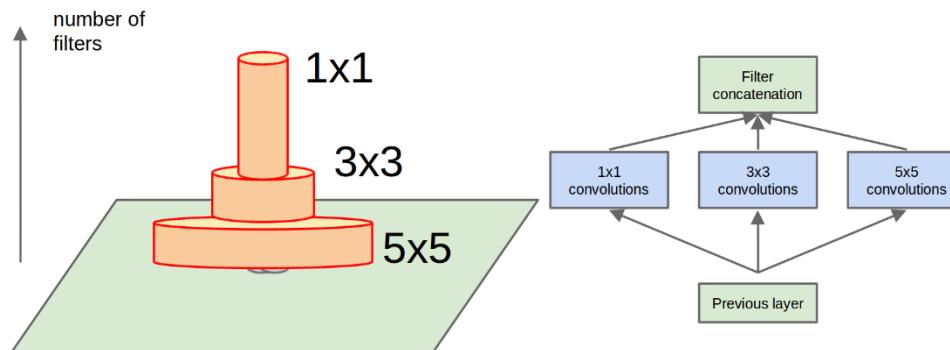


From left: A dog occupying most of the image, a dog occupying a part of it, and a dog occupying very little space (Images obtained from [Unsplash](#)).

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

Sėkmingi tinklai: GoogleNet

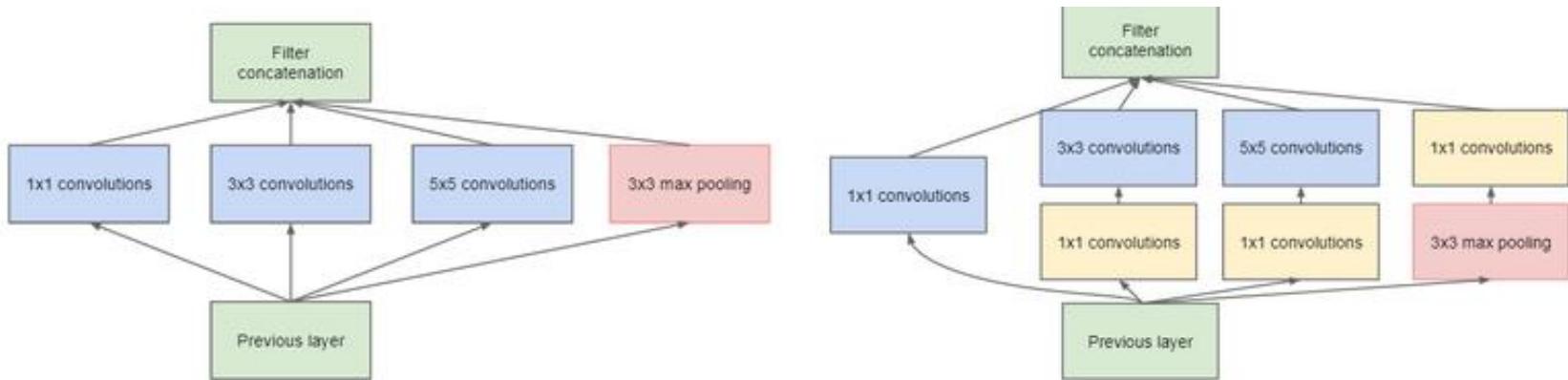
- **GoogleNet / Inception** sluoksnio idėja – padengti didesnį plotą, tačiau išlaikant vaizduose **gerą rezoliuciją** mažam kiekiui informacijos.
- Visų filtrų reikšmės nustatomos **mokymo** metu.



<https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/googlenet.html>

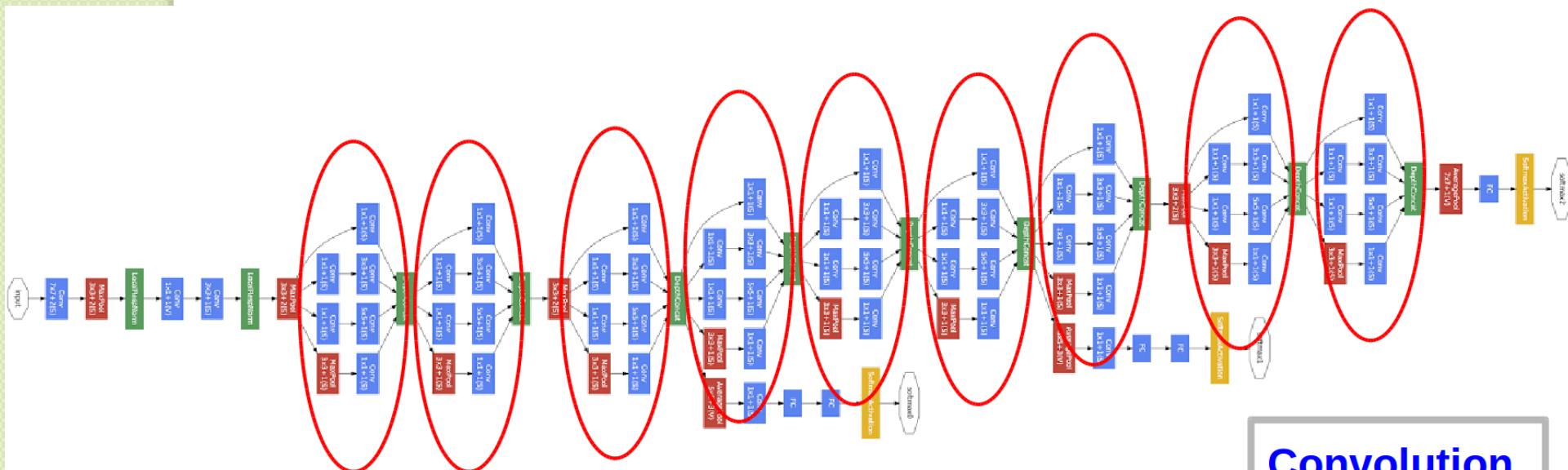
Sėkmingesni tinklai: GoogleNet

- **GoogleNet / Inception** modelyje dar gali būti pridėtas **dimensijos mažinimo** komponentas.



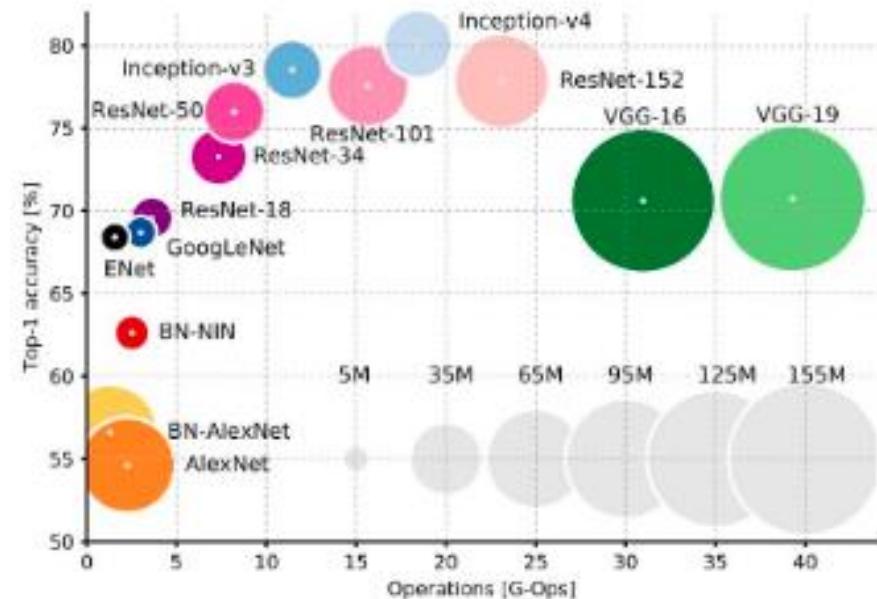
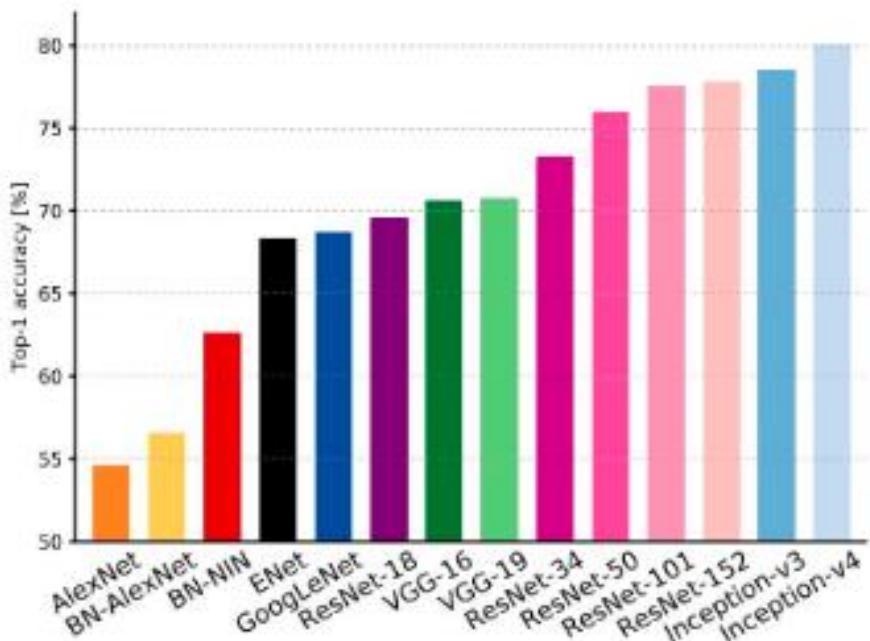
Sėkminges tinklai: GoogleNet

- **GoogleNet/InceptionV1** – konkurso ILSVRC2014 nugalėtojas. Dabar yra **Inception V4** versija.



Convolution
Pooling
Softmax
Concat/Normalize

Gilieji neuroniniai tinklai



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

<https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>

Info

- <https://wiki.tum.de/display/lfdv/Convolutional+Neural+Network+Architectures>
- <http://cs.stanford.edu/people/karpathy/convnets/start.html>
- <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0007-7>
- [https://brohrer.github.io/how convolutional neural networks work.html](https://brohrer.github.io/how_convolutional_neural_networks_work.html)
- <https://cs.stanford.edu/people/karpathy/convnets/demo/mnist.html>