Hugging Face 🤗 Transformers

CPEN 455 Tutorial 8

Felix Fu

UBC
a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

ece
Electrical and Computer Engineering

Reminders

- PA2 has been released. You are about to implement a transformer from scratch.
- The detail of the final course project is posted in this repo.
- Additional leaderboard if you want to get extra points!
- DON'T wait till the last minute to start.

Introduction

- I don’ t know how to code, but I’ d love to experience cutting-edge AI models.🙄
- I don’ t understand the structure of Transformers, but I've been asked to train one to complete a course assignment.😢
- I only have one RTX 3090 GPU, but I want to train a large GPT model to accomplish my tasks.😭

Hugging Face🤗

Tasks Libraries Datasets Languages Licenses Other

Filter Tasks by name

Multimodal

Audio-Text-to-Text | Image-Text-to-Text | Visual Question Answering | Document Question Answering | Video-Text-to-Text | Visual Document Retrieval | Any-to-Any

Computer Vision

Depth Estimation | Image Classification | Object Detection | Image Segmentation | Text-to-Image | Image-to-Text | Image-to-Image | Image-to-Video | Unconditional Image Generation | Video Classification | Text-to-Video | Zero-Shot Image Classification | Mask Generation | Zero-Shot Object Detection | Text-to-3D | Image-to-3D | Image Feature Extraction | Keypoint Detection

Natural Language Processing

Text Classification | Token Classification | Table Question Answering | Question Answering | Zero-Shot Classification | Translation | Summarization | Feature Extraction | Text Generation | Text2Text Generation | Fill-Mask | Sentence Similarity

Audio

Text-to-Speech | Text-to-Audio | Automatic Speech Recognition | Audio-to-Audio | Audio Classification | Voice Activity Detection

Tabular

Tabular Classification | Tabular Regression | Time Series Forecasting

Reinforcement Learning

Reinforcement Learning | Robotics

Other

Graph Machine Learning

Models 1,490,739 Filter by name Full-text search Sort: Trending

Qwen/QwQ-32B
Text Generation • Updated about 2 hours ago • 132k • • 1.81k

deepseek-ai/DeepSeek-R1
Text Generation • Updated 15 days ago • 3.43M • • 11.1k

microsoft/Phi-4-multimodal-instruct
Automatic Speech Recognition • Updated 3 days ago • 303k • 1.06k

Wan-AI/Wan2.1-T2V-14B
Text-to-Video • Updated 12 days ago • 191k • • 967

CohereForAI/aya-vision-8b
Image-Text-to-Text • Updated 6 days ago • 144k • 220

tencent/HunyuanVideo-I2V
Updated 3 days ago • 1.35k • 200

SparkAudio/Spark-TTS-0.5B
Text-to-Speech • Updated 3 days ago • 4.26k • 206

THUDM/CogView4-6B
Text-to-Image • Updated 6 days ago • 8.46k • • 175

allenai/olmOCR-7B-0225-preview
Image-Text-to-Text • Updated 14 days ago • 153k • 518

CohereForAI/aya-vision-32b
Image-Text-to-Text • Updated 6 days ago • 650 • 153

black-forest-labs/FLUX.1-dev
Text-to-Image • Updated Aug 16, 2024 • 2.64M • • 9.26k

bartowski/Qwen_QwQ-32B-GGUF
Text Generation • Updated 5 days ago • 117k • 124

perplexity-ai/r1-1776
Text Generation • Updated 12 days ago • 39.3k • • 2.08k

ASLP-lab/DiffRhythm-base
Updated 5 days ago • 117

Qwen/QwQ-32B-GGUF
Text Generation • Updated about 1 hour ago • 75.2k • 105

lodestones/Chroma
Text-to-Image • Updated about 8 hours ago • 109

hexgrad/Kokoro-82M
Text-to-Speech • Updated 6 days ago • 1.56M • 3.61k

Comfy-Org/Wan_2.1_ComfyUI_repackaged
Updated 3 days ago • 246

microsoft/Phi-4-mini-instruct
Text Generation • Updated 5 days ago • 117k • 326

microsoft/OmniParser-v2.0
Image-Text-to-Text • Updated 20 days ago • 8.92k • 1.13k

GSAI-ML/LLaDA-8B-Instruct
Text Generation • Updated 12 days ago • 19.6k • 198

ElectricAlexis/NotaGen
Updated 12 days ago • 99

Wan-AI/Wan2.1-I2V-14B-720P
Image-to-Video • Updated 12 days ago • 63.3k • 345

Qwen/QwQ-32B-AWQ
Text Generation • Updated about 2 hours ago • 50.8k • 70

Qwen/QwQ-32B-Preview
Text Generation • Updated Jan 11 • 256k • • 1.71k

stabilityai/stable-diffusion-3.5-large
Text-to-Image • Updated Oct 22, 2024 • 161k • • 2.44k

microsoft/Magma-8B
Image-Text-to-Text • Updated 5 days ago • 10.9k • 324

Lightricks/LTX-Video
Text-to-Video • Updated 4 days ago • 350k • • 1.06k

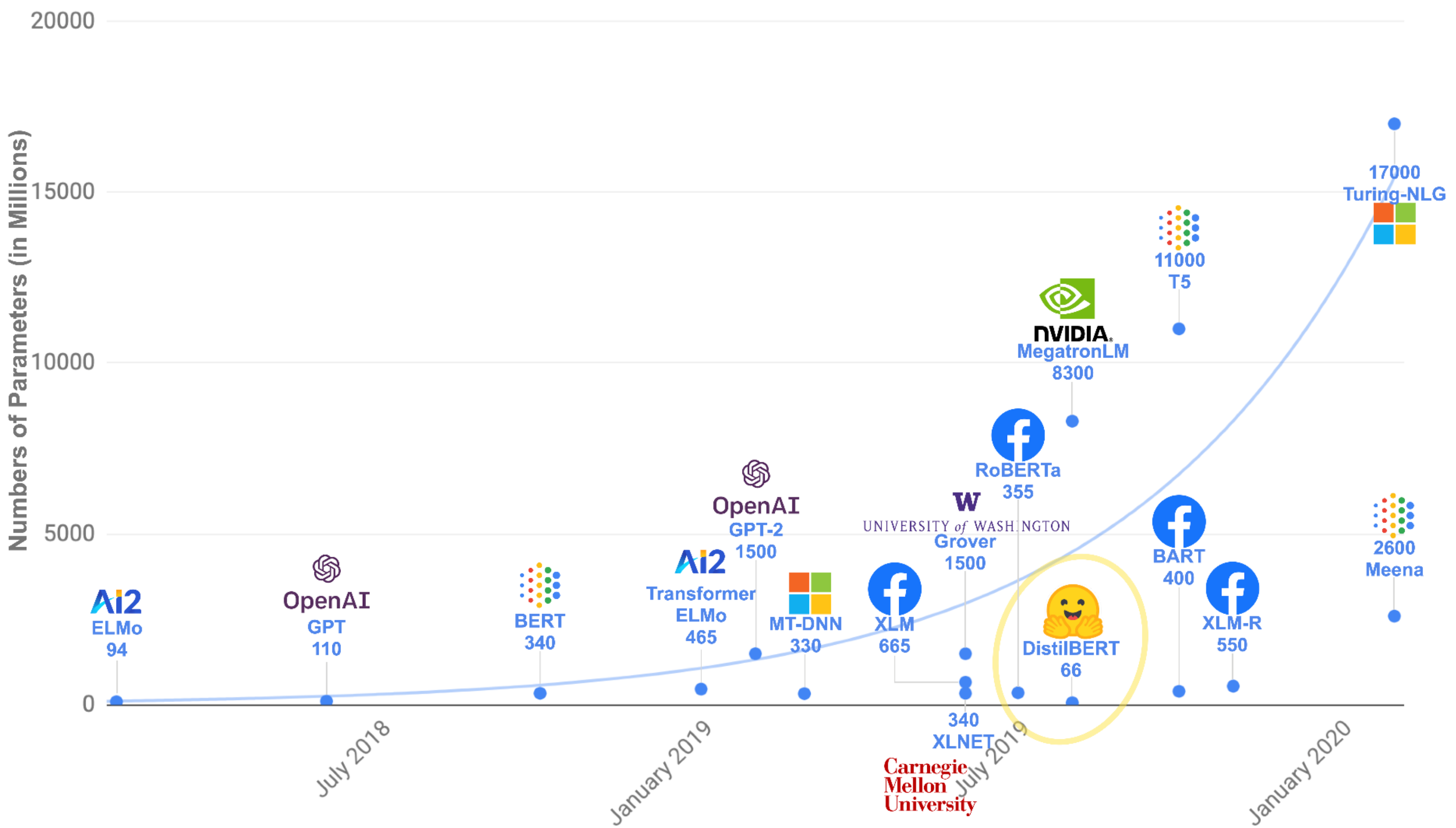
meta-llama/Llama-3.3-70B-Instruct
Text Generation • Updated Dec 21, 2024 • 757k • • 2.12k

agents-course/notebooks
Updated 6 days ago • 227

Previous 1 2 3 ... 100 Next

ical and
uter
eering

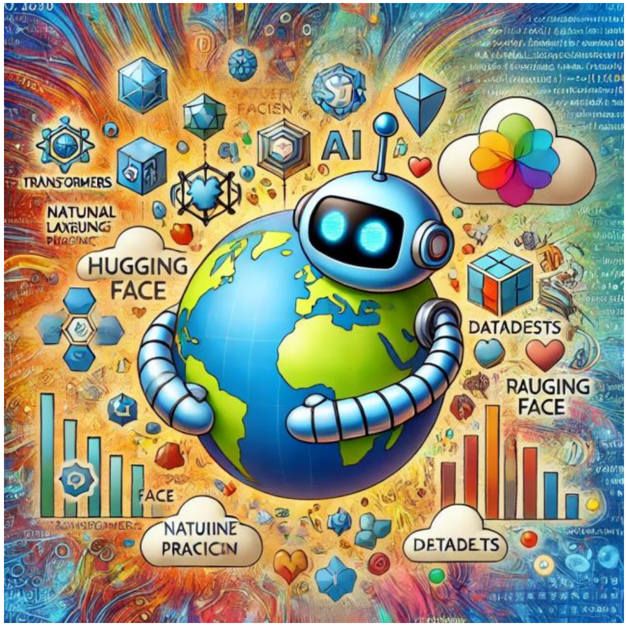
Transformers



Transformers

The 🤗 Transformers library provides a single API through which any Transformer model can be loaded, trained, and saved.

- Ease of use
- Flexibility
- Simplicity



[Chat]() history with GPT4o

Installation

- 🤗 Transformers is tested on Python 3.6+, PyTorch 1.1.0+, TensorFlow 2.0+, and Flax.

Install with pip

```
pip install 'transformers[torch]'
```

Install with Conda

```
conda install conda-forge::transformers
```

Follow the 🤗 instructions:
https://huggingface.co/docs/transformers/en/installation

Basic Usage

- The most basic object in the 🤗 Transformers library is the pipeline() function.

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

Basic Usage

- The most basic object in the 🤗 Transformers library is the pipeline() function.

```
classifier(
    ["I've been waiting for a HuggingFace course my whole life.", "I hate this so much!"]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437},
 {'label': 'NEGATIVE', 'score': 0.9994558095932007}]
```

Task	Description	Modality	Pipeline identifier
Text classification	assign a label to a given sequence of text	NLP	pipeline(task="sentiment-analysis")
Text generation	generate text given a prompt	NLP	pipeline(task="text-generation")
Summarization	generate a summary of a sequence of text or document	NLP	pipeline(task="summarization")
Image classification	assign a label to an image	Computer vision	pipeline(task="image-classification")
Image segmentation	assign a label to each individual pixel of an image (supports semantic, panoptic, and instance segmentation)	Computer vision	pipeline(task="image-segmentation")
Object detection	predict the bounding boxes and classes of objects in an image	Computer vision	pipeline(task="object-detection")
Audio classification	assign a label to some audio data	Audio	pipeline(task="audio-classification")
Automatic speech recognition	transcribe speech into text	Audio	pipeline(task="automatic-speech-recognition")
Visual question answering	answer a question about the image, given an image and a question	Multimodal	pipeline(task="vqa")
Document question answering	answer a question about the document, given a document and a question	Multimodal	pipeline(task="document-question-answering")
Image captioning	generate a caption for a given image	Multimodal	pipeline(task="image-to-text")

Tokenizer

```
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172, 2607,  2026,  28
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,     0,     0,
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

Tokenizer

- Splitting the input into words, subwords, or symbols (like punctuation) that are called tokens.
- Mapping each token to an integer [Token ID].
- Adding additional inputs that may be useful to the model.

Tokenizer

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)

print(tokens)
```

The output of this method is a list of strings, or tokens:

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

```
ids = tokenizer.convert_tokens_to_ids(tokens)

print(ids)
```

```
[7993, 170, 11303, 1200, 2443, 1110, 3014]
```

Models

- 🤗 Transformers provides an AutoModel class which also has a from_pretrained() method:

```
from transformers import AutoModel

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
```

```
outputs = model(**inputs)
print(outputs.last_hidden_state.shape)
```

```
torch.Size([2, 16, 768])
```

Models

- If you know the type of model you want to use, you can use the class that defines its architecture directly.

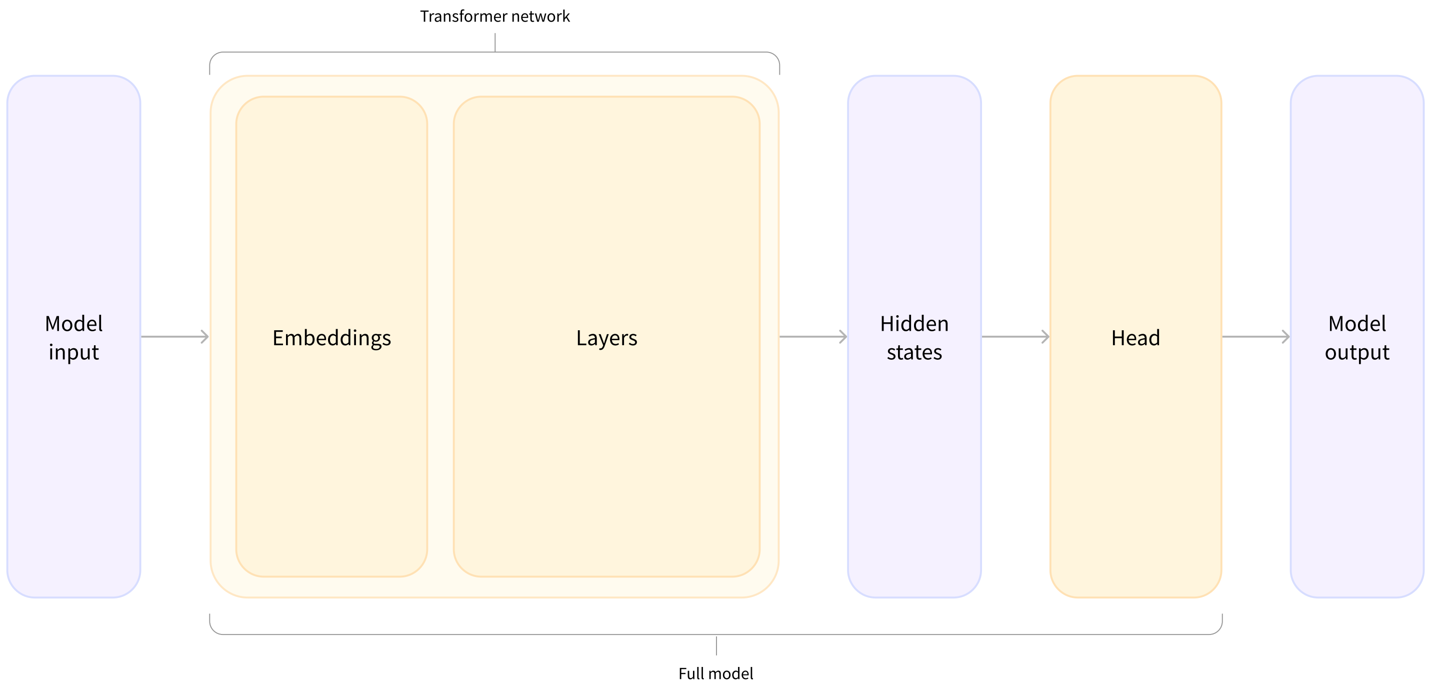
```
from transformers import BertConfig, BertModel

# Building the config
config = BertConfig()

# Building the model from the config
model = BertModel(config)
```

```
print(config)
```

```
BertConfig {
  [...]
  "hidden_size": 768,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  [...]
}
```

Model heads

- The model heads take the high-dimensional vector of hidden states as input and project them onto a different dimension.



Model heads

- The model heads take the high-dimensional vector of hidden states as input and project them onto a different dimension.

```
from transformers import AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = model(**inputs)
```

```
print(outputs.logits.shape)
```

```
torch.Size([2, 2])
```

Model heads

- There are many different architectures available in 🤗 Transformers, with each one designed around tackling a specific task.

  - *Model (retrieve the hidden states)
  - *ForCausalLM
  - *ForMaskedLM
  - *ForMultipleChoice
  - *ForQuestionAnswering
  - *ForSequenceClassification
  - *ForTokenClassification
  - and others 🤗

Postprocessing the output

- Covert logits to probability

```
print(outputs.logits)
```

```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward>)
```

```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

Inference

- Put all these processes to together

```python
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")
output = model(**tokens)
```

Training

- How to train or fine-tune a model using 🤗 Transformers

```python
import torch
from transformers import AdamW, AutoTokenizer, AutoModelForSequenceClassification

# Same as before
checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = [
    "I've been waiting for a HuggingFace course my whole life.",
    "This course is amazing!",
]
batch = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")

# This is new
batch["labels"] = torch.tensor([1, 1])

optimizer = AdamW(model.parameters())
loss = model(**batch).loss
loss.backward()
optimizer.step()
```

Processing the data

- Loading a dataset from the Hub

```
from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")
raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 3668
    })
    validation: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 408
    })
    test: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 1725
    })
})
```

Processing the data

- Pre-processing the data

```python
def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)
```

```python
tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
tokenized_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', '
        num_rows: 3668
    })
    validation: Dataset({
        features: ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', '
        num_rows: 408
    })
    test: Dataset({
        features: ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', '
        num_rows: 1725
    })
})
```

Processing the data

- Get batch!

```
from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```
batch = data_collator(samples)
{k: v.shape for k, v in batch.items()}
```

```
{'attention_mask': torch.Size([8, 67]),
 'input_ids': torch.Size([8, 67]),
 'token_type_ids': torch.Size([8, 67]),
 'labels': torch.Size([8])}
```

Processing the data

- The whole process

```python
from datasets import load_dataset
from transformers import AutoTokenizer, DataCollatorWithPadding

raw_datasets = load_dataset("glue", "mrpc")
checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)


def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)


tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Training

- Specify the training arguments

```python
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

- Define our model

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

Training

- Define a Trainer

```
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

- Start training

```
trainer.train()
```

Evaluation

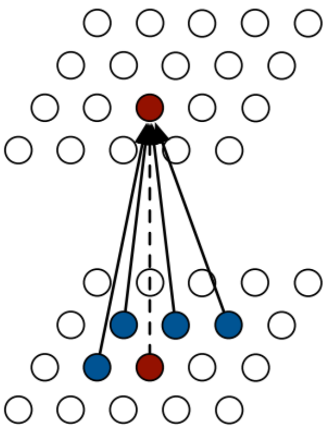
- Define a compute_metrics() function:

```python
def compute_metrics(eval_preds):
    metric = evaluate.load("glue", "mrpc")
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

- Start training

```python
training_args = TrainingArguments("test-trainer", evaluation_strategy="epoch")
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

Pixel CNN



CPEN 455 Tutorial 8

Felix Fu

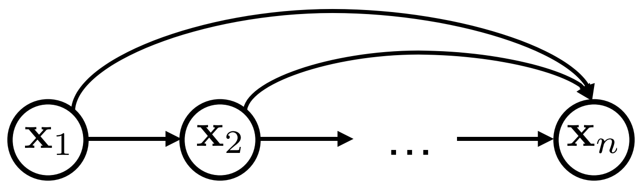
Image Credit: [1]

Autoregressive Models

We are a given n-dimensional data x

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i | \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i | \mathbf{x}_{<i})$$

Graphical model:



PixelCNNs

Autoregressive model for images.

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i|\mathbf{x}_{<i})$$

$\mathbf{x}_i$ is pixel value, e.g., {0, 1, …, 255}

n = height × width

Every term $p_\theta(\mathbf{x}_i|\mathbf{x}_{<i})$ is modeled by the same CNN (softmax readout)

PixelCNNs

$$p_\theta(\mathbf{x}_i | \mathbf{x}_{<i})$$

Conditioned on all pixels that are top-left!

One can also vectorize an image as a sequence and use RNNs to build the autoregressive model, e.g., PixelRNNs [2].

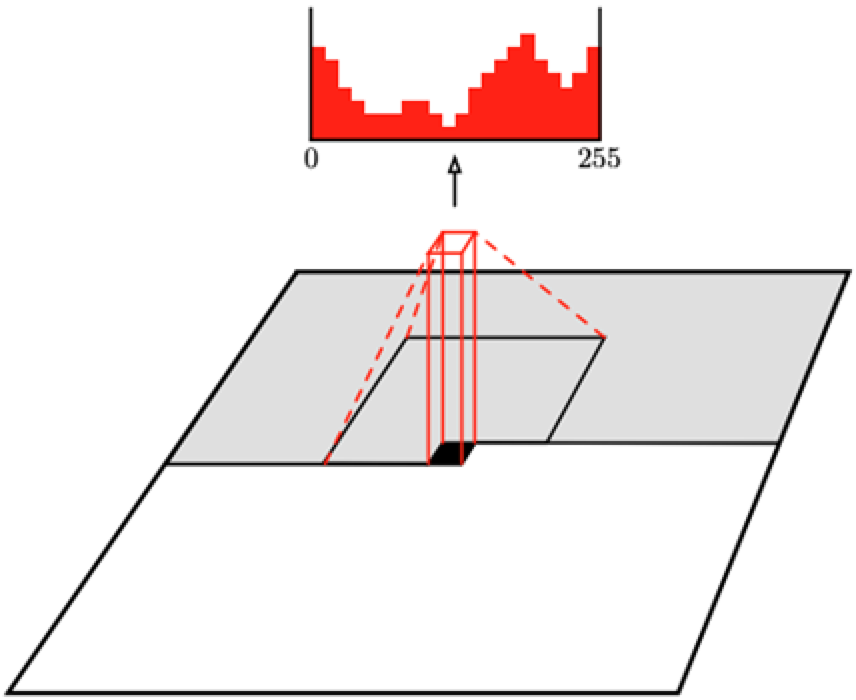


Image Credit: [1]

What About Color Images?

Autoregressive conditioning again along channels:

$$p_\theta(\mathbf{x}_R, \mathbf{x}_G, \mathbf{x}_B) = \prod_i^n p_\theta(x_{R,i} | \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i}) \times$$
$$p_\theta(x_{G,i} | x_{R,i}, \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i}) \times$$
$$p_\theta(x_{B,i} | x_{G,i}, x_{R,i}, \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i})$$

How to Implement?

1. Mask Input
2. Convolution

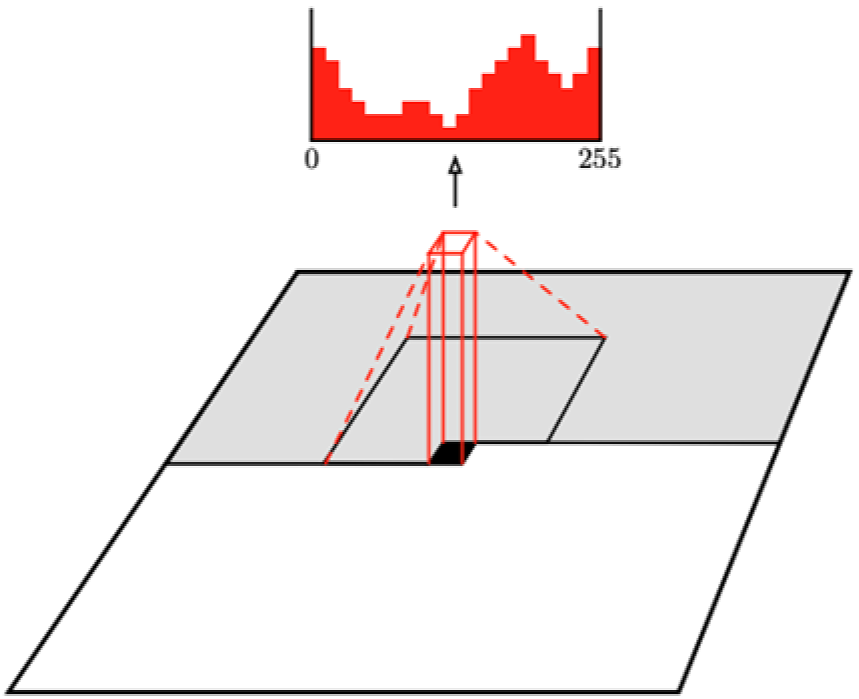


Image Credit: [1]

How to Implement?

For each image, we need $H \times W$ masks and convolutions to compute the likelihood!

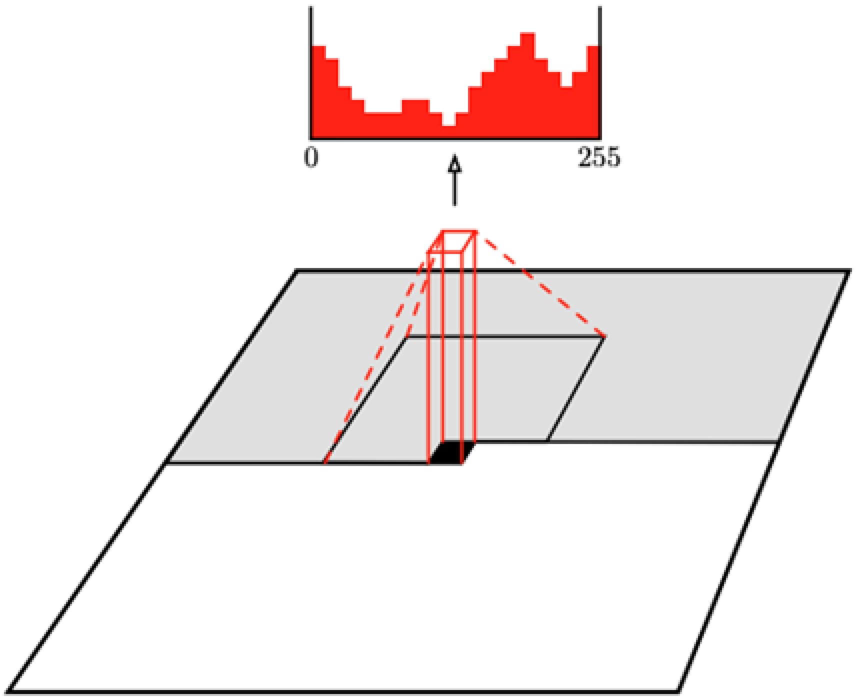
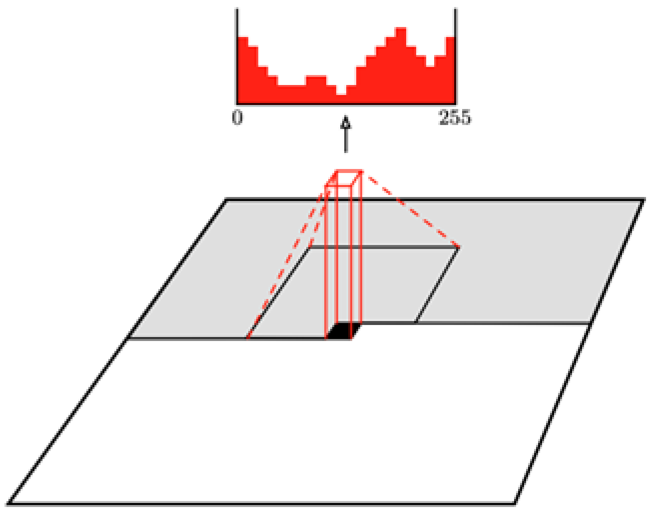


Image Credit: [1]

Solutions in PixelCNNs

Masked Filter + Smart Stack of Regular Convolutions!

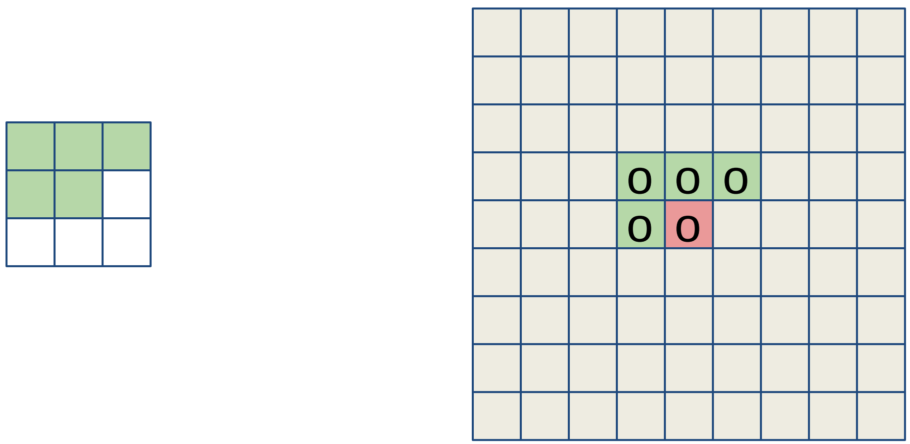


1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Image Credit: [1]

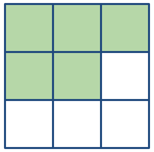
Masked Filter

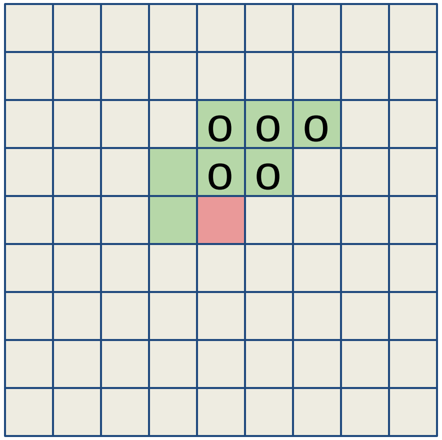
Masked 3×3 filter



Masked Filter

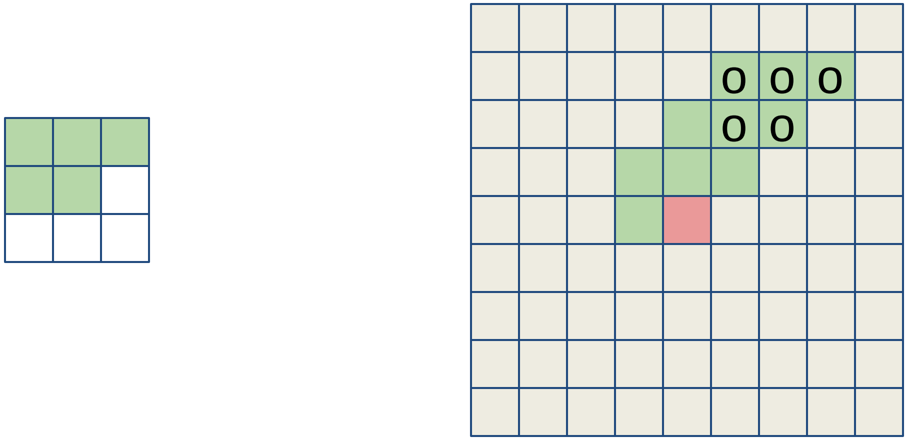
Masked 3 $\times$ 3 filter





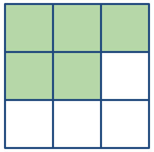
Masked Filter

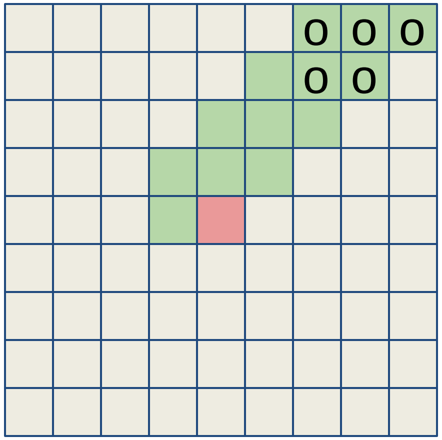
Masked 3×3 filter



Masked Filter

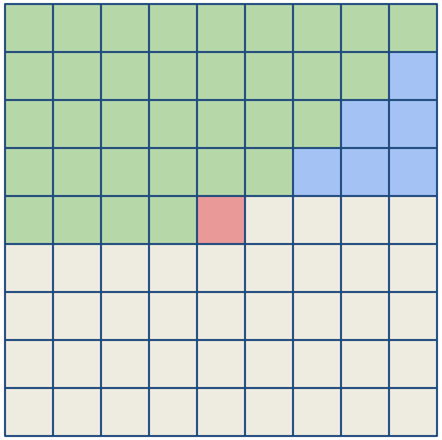
Masked 3 $\times$ 3 filter





Masked Filter

Masked 3×3 filter



Naively applying masked filter causes blind spots (blue area)!

How to Resolve Blind Spots?

Applying two stacks of masked convolutions!

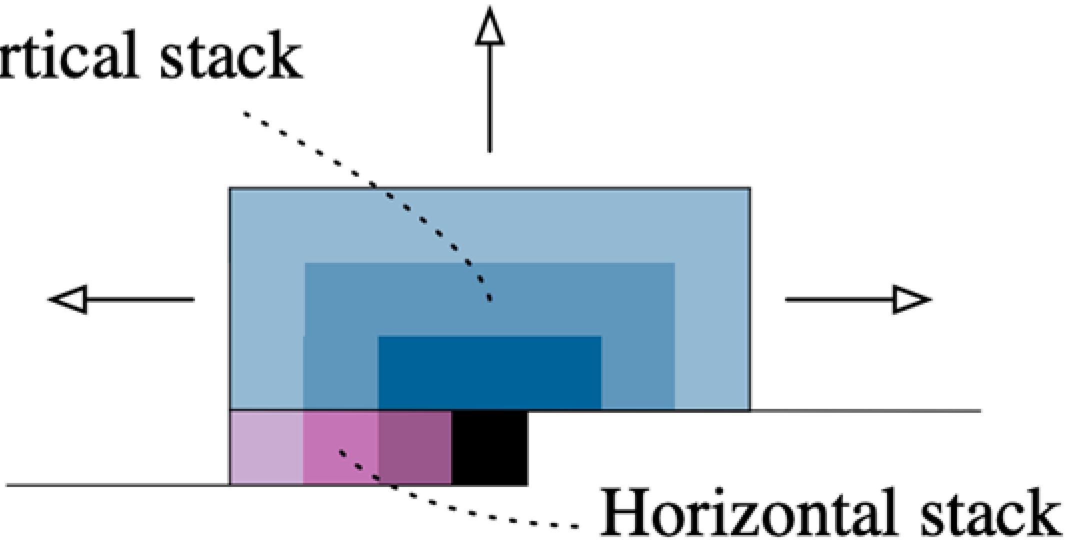
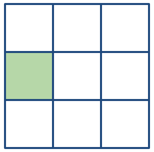


Image Credit: [1]

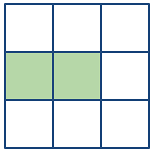
How to Resolve Blind Spots?

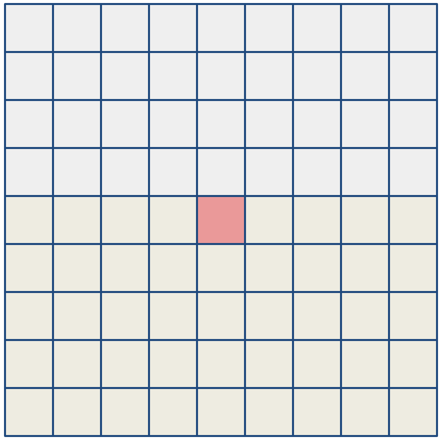
Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



Horizontal Mask 2

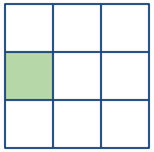


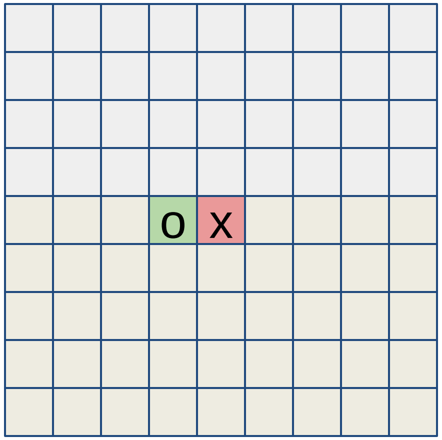


How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



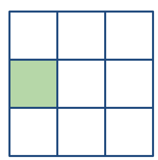


Mask 1 → Mask 2 → ... → Mask 2

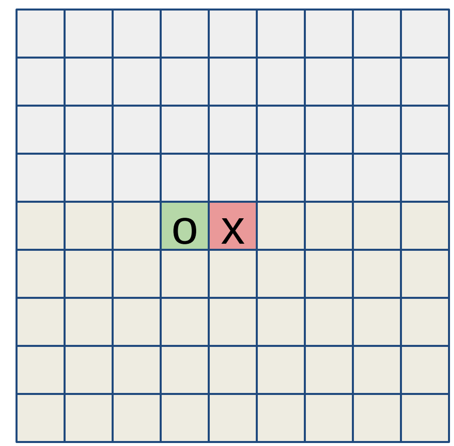
How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



Avoid using information at current location!



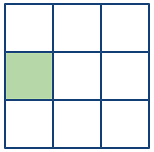
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

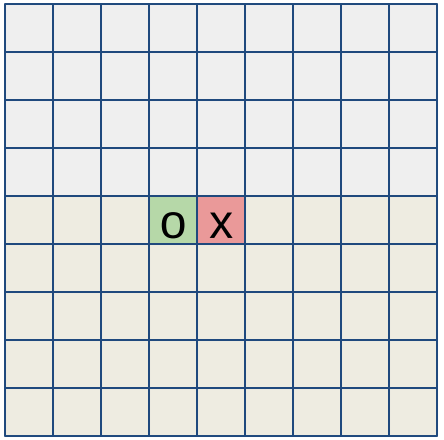
Horizontal Stack (Implemented by masked 2D convolution)

Note that the same masked filter is convolved everywhere!

Horizontal Mask 1



Avoid using information at current location!

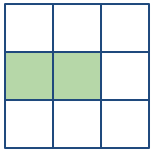


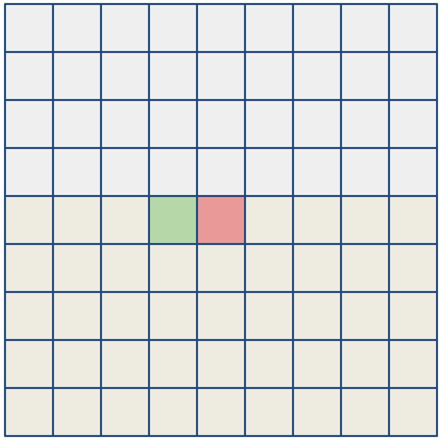
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



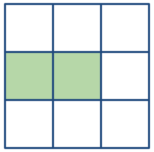


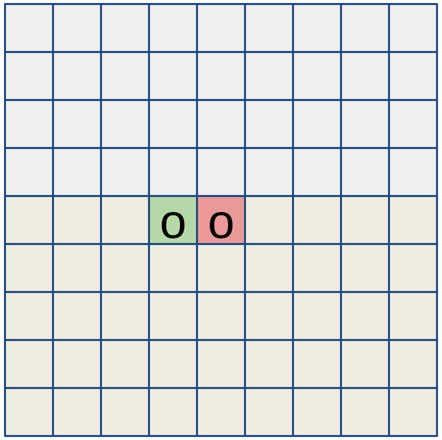
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



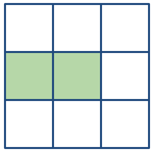


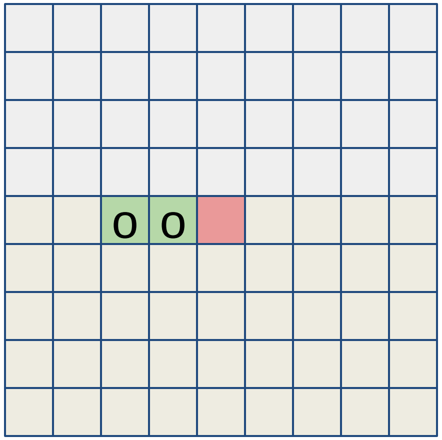
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



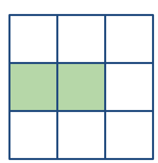


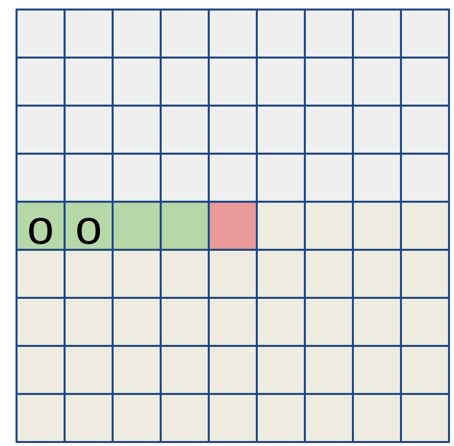
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



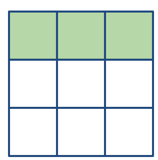


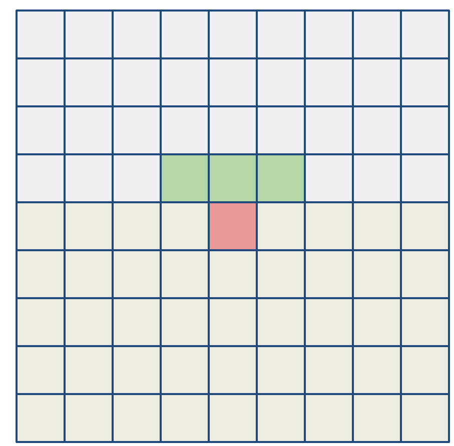
Mask 1 → Mask 2 → ... → **Mask 2**

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1

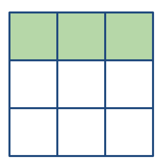




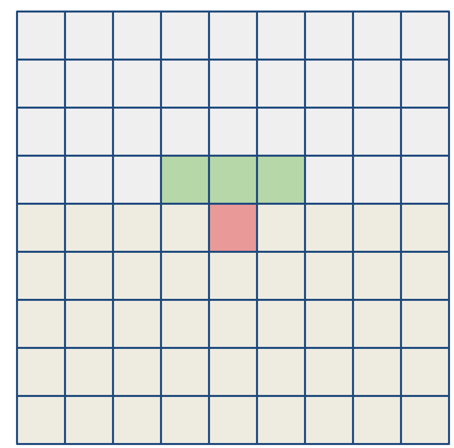
How to Resolve Blind Spots?s

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



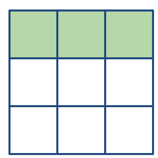
Avoid using information at current location!



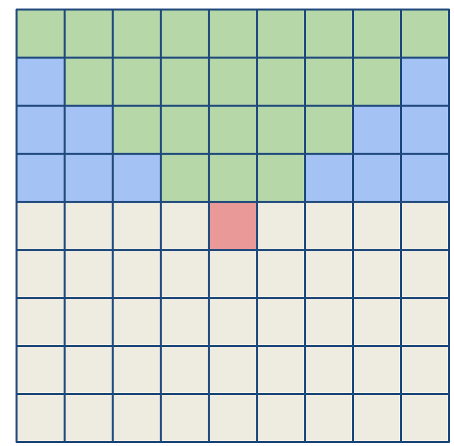
How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



Applying vertical masked filter causes blind spots (blue area) too!

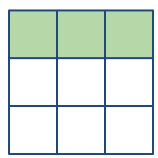


How to Resolve Blind Spots?

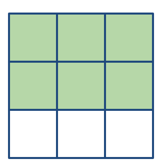
Vertical Stack (Implemented by masked 2D convolution)

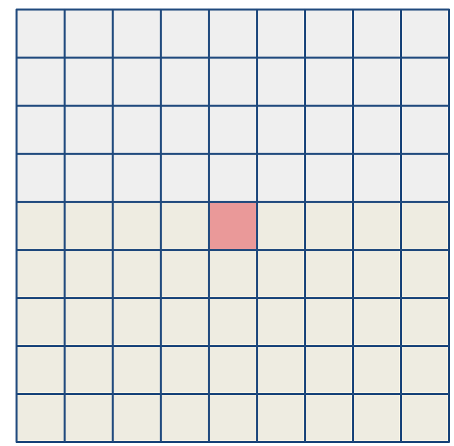
We again use two masked filters to remove blind spots!

Vertical Mask 1



Vertical Mask 2

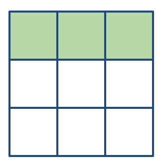


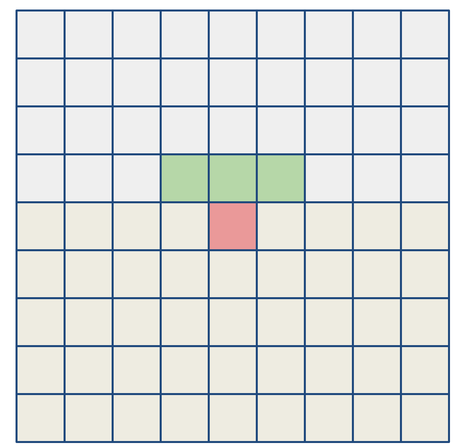


How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1





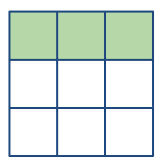
Mask 1 → Mask 2 → ... → Mask 2

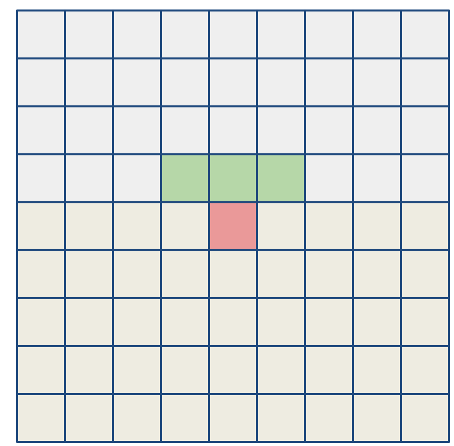
How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Note that the same masked filter is convolved everywhere!

Vertical Mask 1



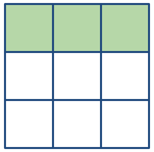


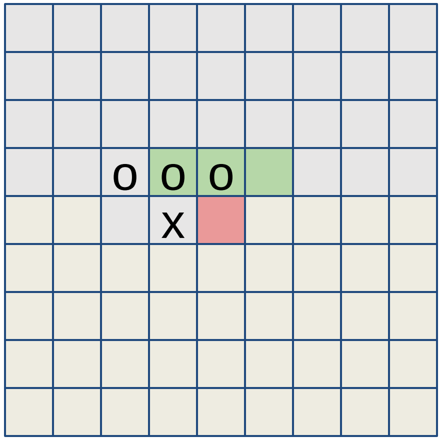
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



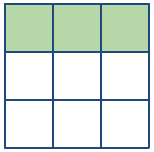


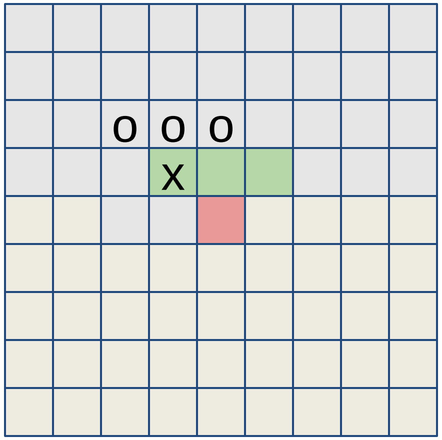
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



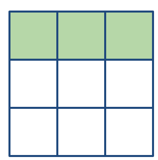


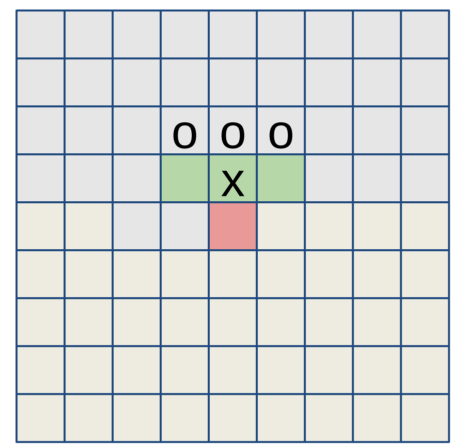
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



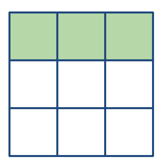


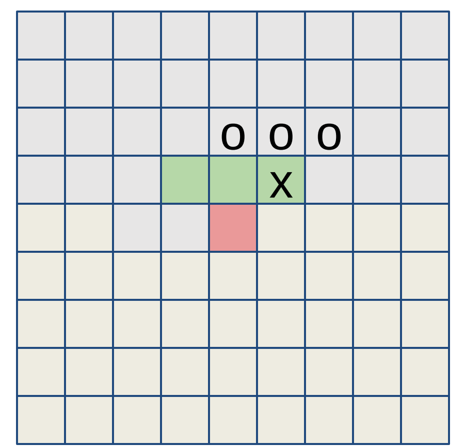
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



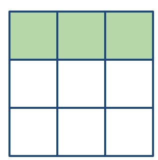


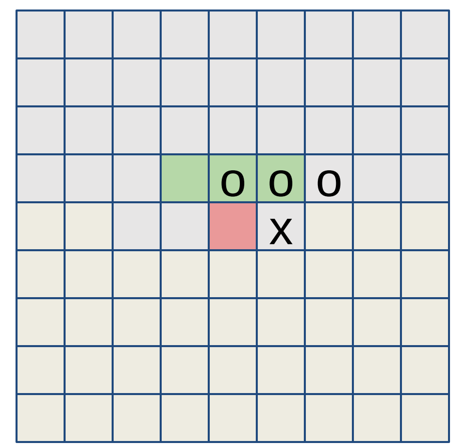
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



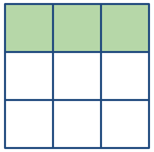


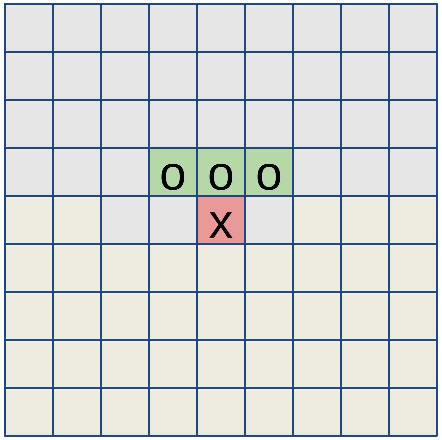
Mask 1 → Mask 2 → … → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



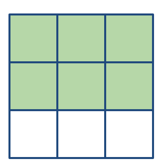


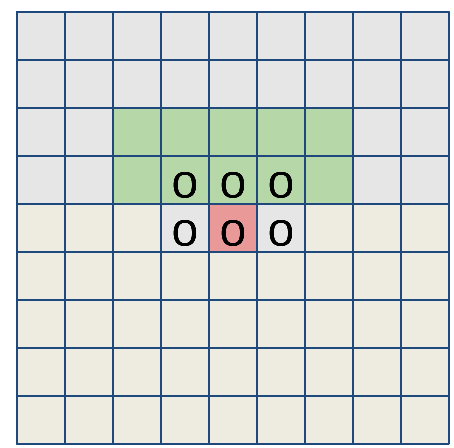
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



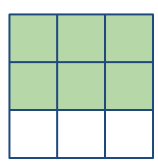


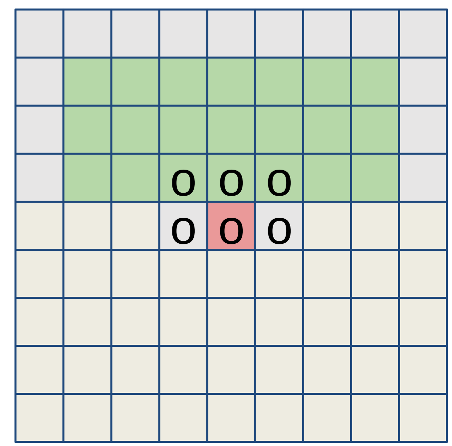
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



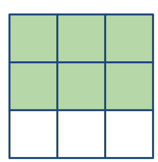


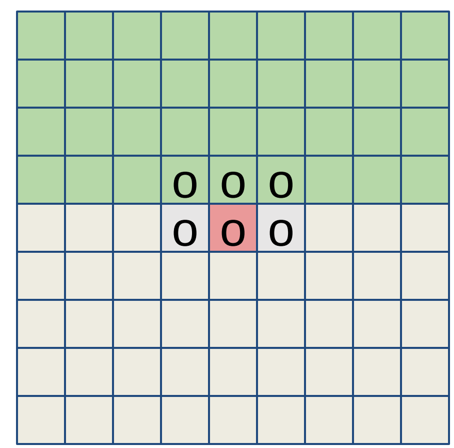
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



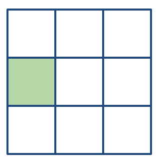


Mask 1 → Mask 2 → ... → **Mask 2**

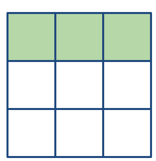
How to Resolve Blind Spots?

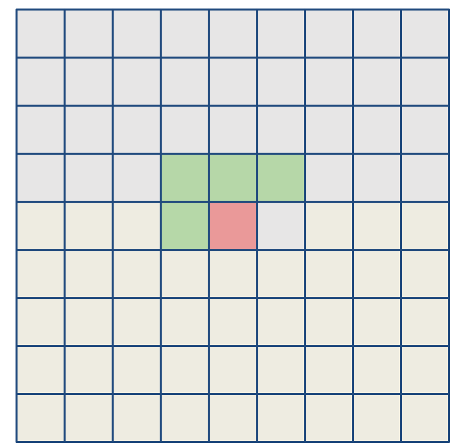
Combine Horizontal and Vertical Stacks

Horizontal Mask 1



Vertical Mask 1



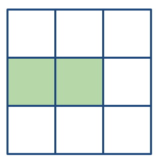


Layer 1 → Layer 2 → ... → Layer L

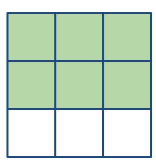
How to Resolve Blind Spots?

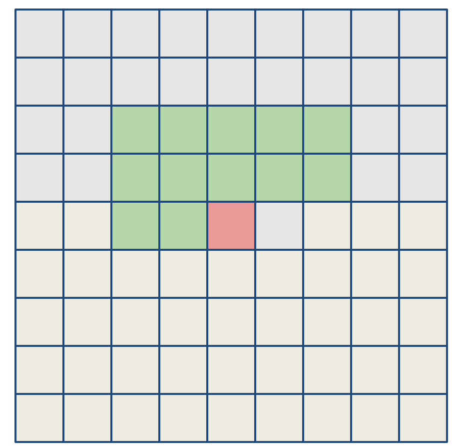
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



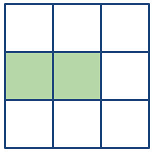


Layer 1 → **Layer 2** → ... → Layer L

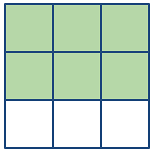
How to Resolve Blind Spots?

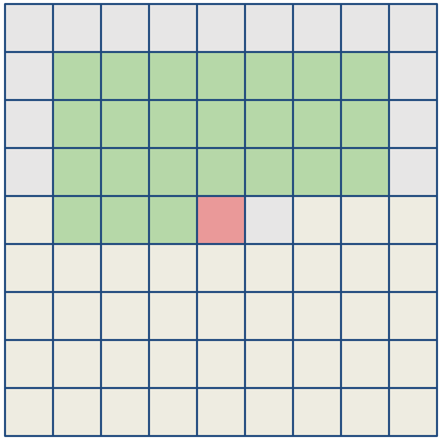
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



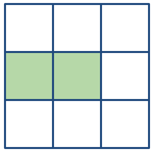


Layer 1 → Layer 2 → ... → Layer L

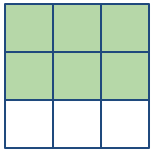
How to Resolve Blind Spots?

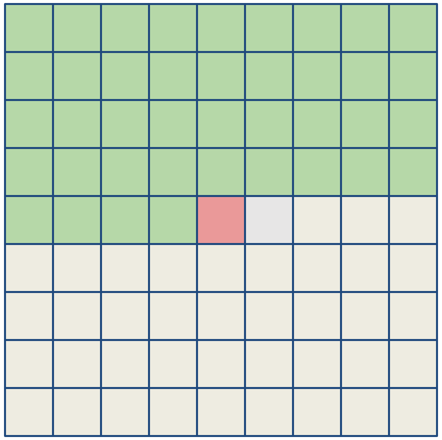
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



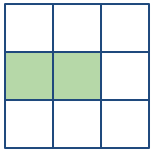


Layer 1 → Layer 2 → ... → **Layer L**

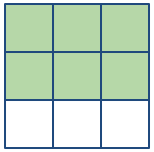
How to Resolve Blind Spots?

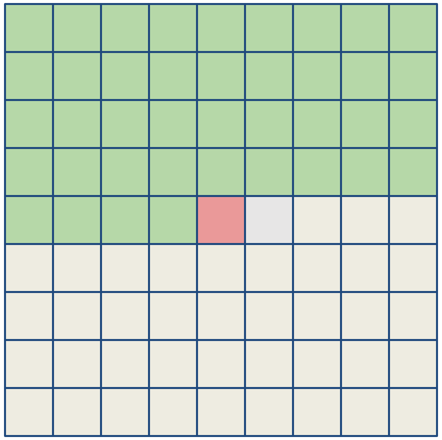
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



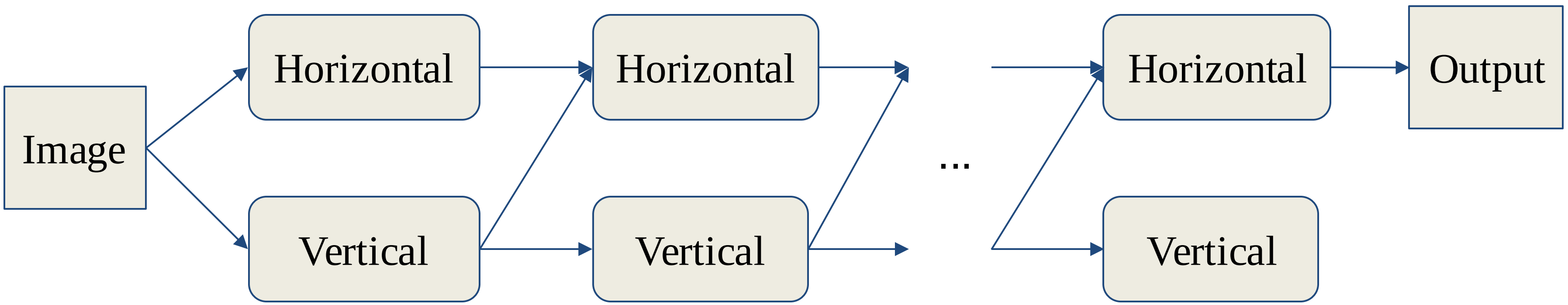
Vertical Mask 2





Layer 1 → Layer 2 → ... → **Layer L**

PixelCNN Process



PixelCNN Architecture

Gated Convolutions $$\mathbf{y} = \tanh\left(\mathbf{W}_f \mathbf{x}\right) \odot \sigma\left(\mathbf{W}_g \mathbf{x}\right)$$

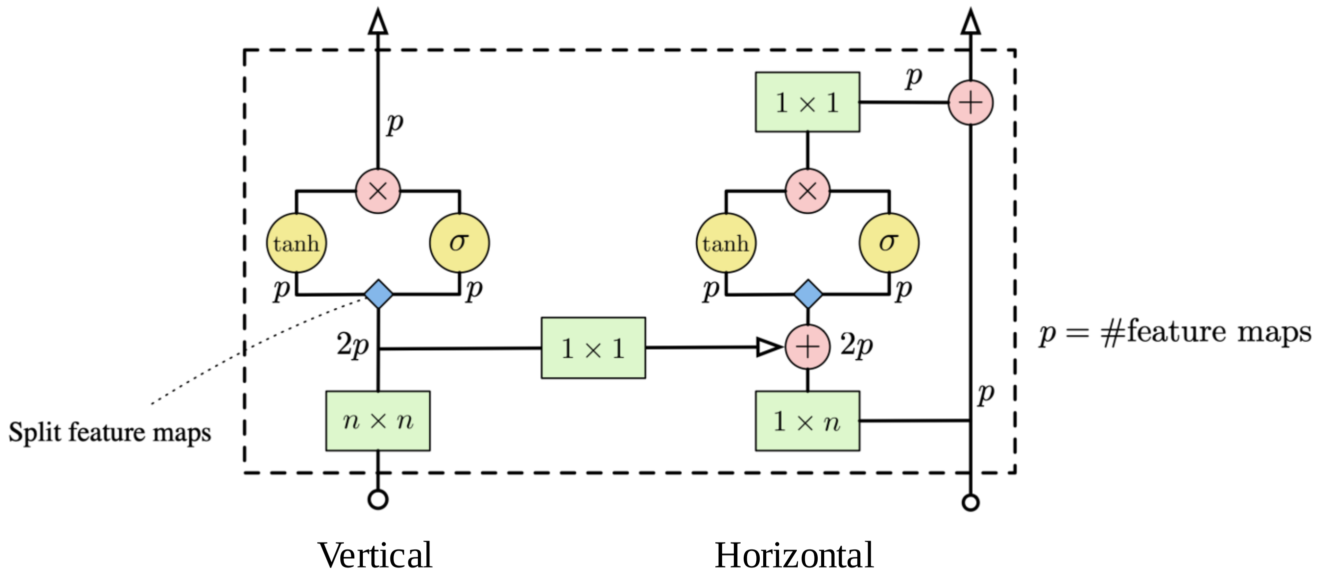


Image Credit: [1]

Pros

+ **Parallel Training**

One forward pass to compute losses at all locations (i.e., all conditional probabilities)!

+ **Strong Performances**

PixelCNN++ [3] further improves performances by:

1. Softmax → discretized mixture of logistic distributions
2. Downsample & upsample, dropout, skip connections, etc.

$$\nu \sim \sum_{i=1}^{K} \pi_i \mathrm{logistic}(\mu_i, s_i)$$

$$P(x|\pi,\mu,s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x+0.5-\mu_i)/s_i) - \sigma((x-0.5-\mu_i)/s_i)\right],$$

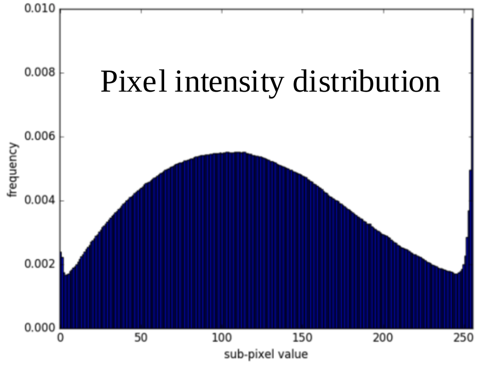


Image Credit: [3]

Pros vs Cons

+ **Parallel Training**

One forward pass to compute losses at all locations (i.e., all conditional probabilities)!

+ **Strong Performances**

PixelCNN++ [3] further improves performances by:

1. Softmax → discretized mixture of logistic distributions
2. Downsample & upsample, dropout, skip connections, etc.

- Slow Sampling

This is due to the sequential nature of autoregressive sampling.
It could be further improved by methods, e.g., [5].

References

Hugging Face 🤗 : https://huggingface.co/

Transformers 🤗 : https://huggingface.co/docs/transformers/en/index

More courses 🤗: https://huggingface.co/learn/nlp-course

[1] van den Oord, A., et al. "Conditional Image Generation with PixelCNN Decoders." In Advances in Neural Information Processing Systems 29, pp. 4790–4798 (2016).

[2] van den Oord, A., et al. "Pixel Recurrent Neural Networks." arXiv preprint arXiv:1601.06759 (2016).

[3] Salimans, Tim, et al. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications." arXiv preprint arXiv:1701.05517 (2017).

[4] https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial12/Autoregressive_Image_Modeling.html

[5] Song, Y., Meng, C., Liao, R. and Ermon, S., 2021, July. Accelerating feedforward computation via parallel nonlinear equation solving. In International Conference on Machine Learning (pp. 9791-9800). PMLR.

[6] https://en.wikipedia.org/wiki/Logistic_distribution