Aplicação de redes neurais convolucionais para a classificação multirrótulo de peças de roupa

EEL7513 – Projeto Final
2019.2

Kauê Cano Souza – 18204680
Ruan Cardoso Comelli – 201901993

CONTEÚDO

INTRODUÇÃO

Com o crescimento do *e-commerce*, marcas de roupa buscam cada vez mais dados para embasar suas decisões.

Utilizando sistemas classificadores *multi-label* de imagens de vestimentas, é expandida a quantidade de informação relacionada a cada produto.

Isso agrega valor tanto para aplicações internas quanto para o monitoramento de competidores.

OBJETIVOS

Construir um modelo empregando mecanismos de atenção para extração de landmarks, categoria e atributos.

Analisar o impacto de diversos parâmetros de treinamento e redes-base na categorização de múltiplos atributos.

OBJETIVO I

Desenvolver um modelo classificador
multi-label seguindo moldes do
estado-da-arte.

Analisar se isso é possível com as limitações
temporais e de recursos.

OBJETIVO 2

Desenvolver um modelo classificador multi-label simplificado.

Analisar resultados obtidos.

Dataset: DeepFashion

Y-Back_Halter_Dress

800 000 imagens classificadas
50 categorias humanamente adquiridas
1000 atributos descritivos
4 a 8 atributos por imagem

Back_Halter_Long_Sleeve
C: 6
LM: 1 0 146 102 ...
A: -1 -1 -1 -1 -1 -1 1 ...

Subset: Attribute Prediction

289 222 imagens classificadas
+ Fashion Landmark Detection Benchmark

REDES

VGG16
2014

ResNet50
2015

MobileNetV2
2018

RECURSOS

NVIDIA P100

GCP - 3,2k usd/mês

NVIDIA V100

GCP - 10k usd/mês

Google Colab

GCP - Free

METODOLOGIA

Tarefas que exigem muitos recursos computacionais - tempo **e** GPUs.

Dataset muito grande - imagens **e** arquivos.

Redes relativamente complexas.

DATA AUGMENTATION

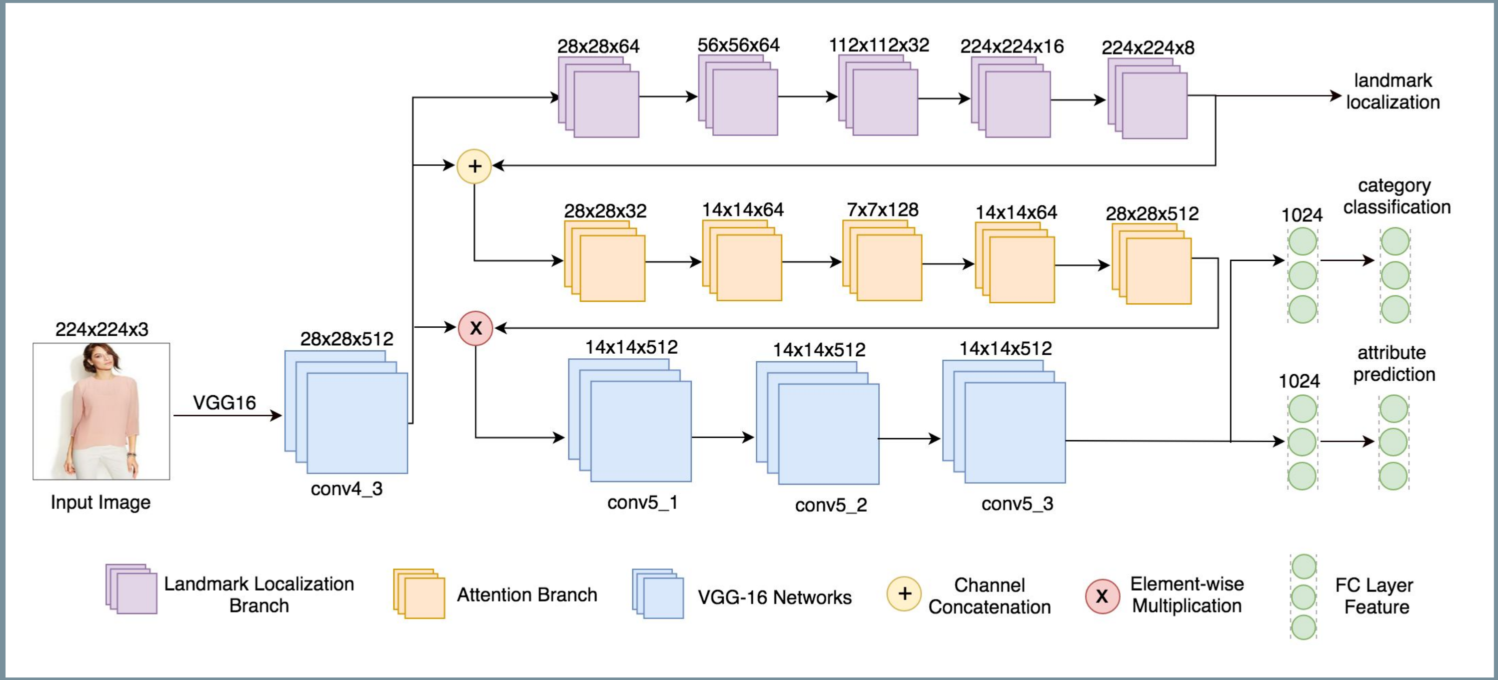
RECURSOS

Nome	GPU	Núcleos CPU	RAM	Disco	Sistema Operacional
nvidia1	Tesla V100	8	26 GB	32 GB SSD	Ubuntu 18.04
nvidia2	Tesla P100	16	40 GB	32 GB SSD	Ubuntu 18.04
cpu1	-	8	20 GB	32 GB SSD	Debian 10.2

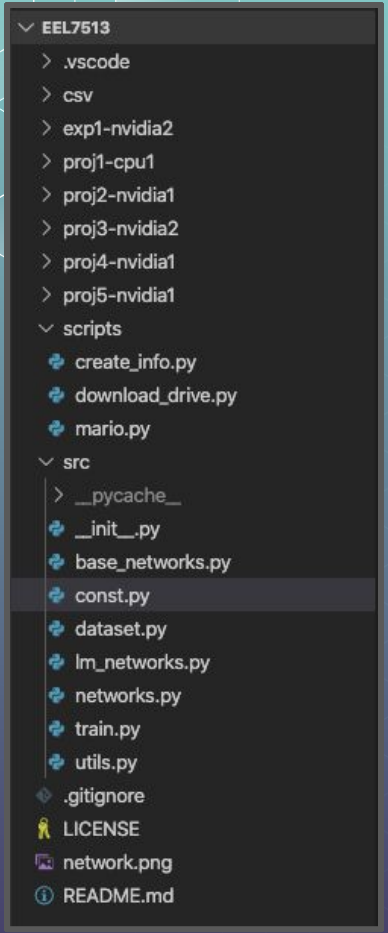
Processor	SMs	CUDA Cores	Tensor Cores	Frequency	TFLOPs (double)	TFLOPs (single)	TFLOPs (half/Tensor)	Cache	Max. Memory	Memory B/W
Nvidia P100 PCIe (Pascal)	56	3,584	N/A	1,126 MHz	4.7	9.3	18.7	4 MB L2	16 GB	720 GB/s
Nvidia V100 PCIe (Volta)	80	5,120	640	1.53 GHz	7	14	112	6 MB L2	16 GB	900 GB/s

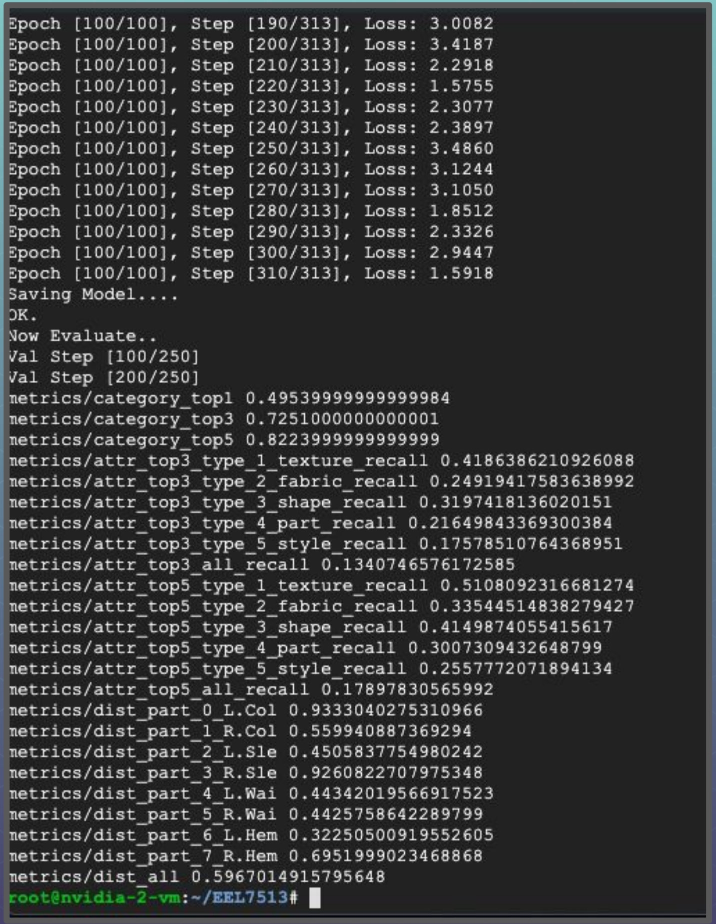
PYTORCH: P-Net

ARQUITETURA



PROJETO





```
Epoch [100/100], Step [190/313], Loss: 3.0082
Epoch [100/100], Step [200/313], Loss: 3.4187
Epoch [100/100], Step [210/313], Loss: 2.2918
Epoch [100/100], Step [220/313], Loss: 1.5755
Epoch [100/100], Step [230/313], Loss: 2.3077
Epoch [100/100], Step [240/313], Loss: 2.3897
Epoch [100/100], Step [250/313], Loss: 3.4860
Epoch [100/100], Step [260/313], Loss: 3.1244
Epoch [100/100], Step [270/313], Loss: 3.1050
Epoch [100/100], Step [280/313], Loss: 1.8512
Epoch [100/100], Step [290/313], Loss: 2.3326
Epoch [100/100], Step [300/313], Loss: 2.9447
Epoch [100/100], Step [310/313], Loss: 1.5918
Saving Model....
OK.
Now Evaluate..
Val Step [100/250]
Val Step [200/250]
metrics/category_top1 0.4953999999999984
metrics/category_top3 0.7251000000000001
metrics/category_top5 0.8223999999999999
metrics/attr_top3_type_1_texture_recall 0.4186386210926088
metrics/attr_top3_type_2_fabric_recall 0.24919417583638992
metrics/attr_top3_type_3_shape_recall 0.3197418136020151
metrics/attr_top3_type_4_part_recall 0.21649843369300384
metrics/attr_top3_type_5_style_recall 0.17578510764368951
metrics/attr_top3_all_recall 0.1340746576172585
metrics/attr_top5_type_1_texture_recall 0.5108092316681274
metrics/attr_top5_type_2_fabric_recall 0.33544514838279427
metrics/attr_top5_type_3_shape_recall 0.4149874055415617
metrics/attr_top5_type_4_part_recall 0.3007309432648799
metrics/attr_top5_type_5_style_recall 0.2557772071894134
metrics/attr_top5_all_recall 0.17897830565992
metrics/dist_part_0_L.Col 0.9333040275310966
metrics/dist_part_1_R.Col 0.559940887369294
metrics/dist_part_2_L.Sle 0.4505837754980242
metrics/dist_part_3_R.Sle 0.9260822707975348
metrics/dist_part_4_L.Wai 0.44342019566917523
metrics/dist_part_5_R.Wai 0.4425758642289799
metrics/dist_part_6_L.Hem 0.32250500919552605
metrics/dist_part_7_R.Hem 0.6951999023468868
metrics/dist_all 0.5967014915795648
root@nvidia-2-vm:~/EEL7513#
```

CONSTANTES

```
# Network
USE_NET = _net
LM_SELECT_VGG = 'conv4_3'
LM_SELECT_VGG_SIZE = 28
LM_SELECT_VGG_CHANNEL = 512
LM_BRANCH = _lm_branch
EVALUATOR = _evaluator
#################

#DATASET SIZE
TRAIN_SPLIT_LEN = 10000
VAL_SPLIT_LEN = 10000


#BATCHES
BATCH_SIZE = 32
VAL_BATCH_SIZE = 40
```

```
#WORK
NUM_WORKERS = 16
NUM_EPOCH = 20

#LR
LEARNING_RATE = 0.0001
LEARNING_RATE_DECAY = 0.8

# LOSS WEIGHT
WEIGHT_LOSS_CATEGORY = 0.01
WEIGHT_LOSS_ATTR = 20
WEIGHT_LOSS_LM_POS = 0.01


# 0-1 WEIGHT
WEIGHT_ATTR_NEG = 0.001
WEIGHT_ATTR_POS = 1
WEIGHT_LANDMARK_VIS_NEG = 0.5
WEIGHT_LANDMARK_VIS_POS = 0.5
```

BASE NET

```python
class CustomUnetGenerator(nn.Module):
    def __init__(self, input_nc, output_nc, num_downs, ngf=64,
                 norm_layer=nn.BatchNorm2d, use_dropout=False, last_act='sigmoid'):
        super(CustomUnetGenerator, self).__init__()
```

```python
class BaseLoss(ModuleWithAttr):

    def __init__(self):
        super(BaseLoss, self).__init__()
        self.category_loss_func = torch.nn
        self.attr_loss_func = torch.nn.Cro
        self.lm_vis_loss_func = torch.nn.C
        self.lm_pos_loss_func = torch.nn.M

    def cal_loss(self, sample, output):
        category_loss = self.category_loss
```

LANDMARKS

```python
class LandmarkBranchUpsample(nn.Module):

    def __init__(self, in_channel=256):
        super(LandmarkBranchUpsample, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, 64, 1, 1, 0)
        self.conv2 = nn.Conv2d(64, 64, 3, 1, 1)
        self.conv3 = nn.Conv2d(64, 64, 3, 1, 1)
        self.conv4 = nn.Conv2d(64, 128, 3, 1, 1)
        self.upconv1 = nn.ConvTranspose2d(128, 64, 4, 2, 1)
        self.conv5 = nn.Conv2d(64, 64, 3, 1, 1)
        self.conv6 = nn.Conv2d(64, 64, 3, 1, 1)
        self.upconv2 = nn.ConvTranspose2d(64, 32, 4, 2, 1)
        self.conv7 = nn.Conv2d(32, 32, 3, 1, 1)
        self.conv8 = nn.Conv2d(32, 32, 3, 1, 1)
        self.upconv3 = nn.ConvTranspose2d(32, 16, 4, 2, 1)
        self.conv9 = nn.Conv2d(16, 16, 3, 1, 1)
        self.conv10 = nn.Conv2d(16, 8, 1, 1, 0)
```

UTILS

```python
class Evaluator(object):

    def __init__(self, category_topk=(1, 3,
        self.category_topk = category_topk
        self.attr_topk = attr_topk
        self.reset()
        with open(const.base_path + 'Anno/l
            ret = []
            f.readline()
            f.readline()
```

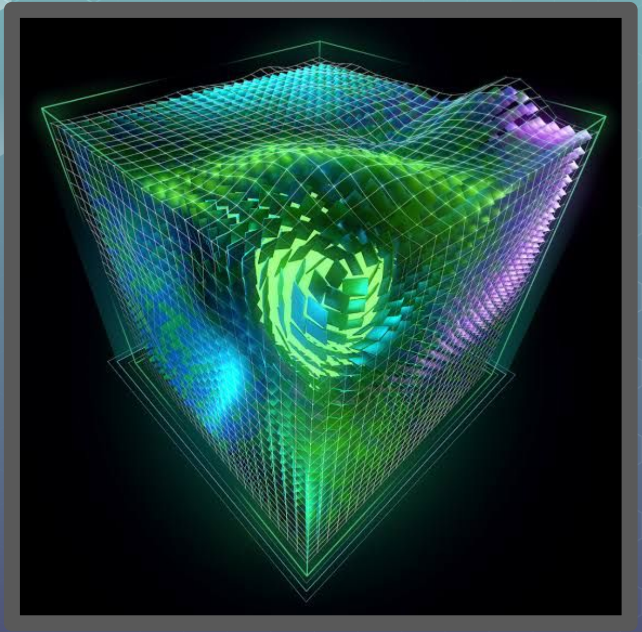
```python
class LandmarkEvaluator(object):

    def __init__(self):

        self.reset()

    def reset(self):
        self.lm_vis_count_all = np.array([0.] * 8)
        self.lm_dist_all = np.array([0.] * 8)

    def landmark_count(self, output, sample):
        if hasattr(const, 'LM_EVAL_USE') and const.LM_EVAL_USE ==
            mask_key = 'landmark_in_pic'
        else:
            mask_key = 'landmark_vis'
        landmark_vis_count = sample[mask_key].cpu().numpy().sum(ax
        landmark_vis_float = torch.unsqueeze(sample[mask_key].floa
```

NETWORK

```
class WholeNetwork(ModuleWithAttr):

    def __init__(self):
        super(WholeNetwork, self).__init__()
        self.vgg16_extractor = VGG16Extractor()
        self.lm_branch = const.LM_BRANCH(const.LM_SELECT_VGG_CHANNEL)
        self.downsample = nn.Upsample((28, 28), mode='bilinear', align_corners=False)
        self.attention_pred_net = CustomUnetGenerator(512 + 1, 512, num_downs=2, ngf=32, last_act='tanh')
        self.pooled_4 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv5_1 = nn.Conv2d(512, 512, 3, padding=1)
        self.conv5_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.conv5_3 = nn.Conv2d(512, 512, 3, padding=1)
        conv5_para_vgg16 = [
            self.vgg16_extractor.vgg[-7].state_dict(),
            self.vgg16_extractor.vgg[-5].state_dict(),
            self.vgg16_extractor.vgg[-3].state_dict(),
        ]
        self.conv5_1.load_state_dict(conv5_para_vgg16[0])
        self.conv5_2.load_state_dict(conv5_para_vgg16[1])
        self.conv5_3.load_state_dict(conv5_para_vgg16[2])
        self.pooled_5 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.category_fc1 = nn.Linear(512 * 7 * 7, 1024)
        self.category_fc2 = nn.Linear(1024, 48)
        self.attr_fc1 = nn.Linear(512 * 7 * 7, 1024)
        self.attr_fc2 = nn.Linear(1024, 1000 * 2)

        self.category_loss_func = torch.nn.CrossEntropyLoss()
        self.attr_loss_func = torch.nn.CrossEntropyLoss(weight=torch.tensor([const.WEIGHT_ATTR_NEG, const.WEIG
```

TRAIN

```python
net = const.USE_NET()
net = net.to(const.device)

learning_rate = const.LEARNING_RATE
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

writer = SummaryWriter(const.TRAIN_DIR)

total_step = len(train_dataloader)
step = 0
for epoch in range(const.NUM_EPOCH):
    net.train()
    for i, sample in enumerate(train_dataloader):
        step += 1
        for key in sample:
            sample[key] = sample[key].to(const.device)
        output = net(sample)
        loss = net.cal_loss(sample, output)

        optimizer.zero_grad()
        loss['all'].backward()
        optimizer.step()
```



CUDA

CUPy

Numpy

+ .cpu() + .cuda()

CASOS

Nome	Máquina	N	Rede *Landmark*	w_N	Épocas	Batch Size	Workers
exp1	nvidia2	50,000	Relevada	100	100	32	16
proj1	cpu1	10,000	Presente	10	20	16	4
proj2	nvidia1	10,000	Relevada	1000	20	32	16
proj3	nvidia2	10,000	Presente	100	20	32	10
proj4	nvidia1	20,000	Presente	100	20	40	14
proj5	nvidia1	50,000	Presente	200	20	50	10

KERAS: K-Net

UTILS

```
def missing_elements(int_list): # source: adapted from <https://stackoverflow.
    int_list = sorted(int_list)
    if int_list:
        start, end = int_list[0], int_list[-1]
        full_list = set(range(start, end + 1))
        return sorted(full_list.difference(int_list))
    else:
        return set([])

def merge_dicts(*dict_args):
    """
    Given any number of dicts, shallow copy and merge into a new dict,
    precedence goes to key value pairs in latter dicts.
    """
    result = {}
    for dictionary in dict_args:
        result.update(dictionary)
    return result

def extract_value(dicts, key, default_behaviour='value', default=None):
    if isinstance(dicts, dict):
        dicts = [dicts]
```

MODEL MANAGER

```python
from pathlib import Path
import json

class ModelManager:
    def __init__(
        self,
        models_path=None,
        table_path=None,
        encoding='utf-8',
        load_table=True,
        file_name_fmt='{index}.data',
        creator_method=None,
        creator_name=None,
        save_method=None,
        load_method=None
    ):
        def default_save_method(model, path):
            import pickle
            with path.open('wb') as file:
                pickle.dump(model, file, protocol=pickle.HIGHEST_PROTOCOL)

        def default_load_method(path):
            import pickle
            with path.open('rb') as file:
                return pickle.load(file)

        if models_path is None:
            models_path = Path('.') / 'models'
        self.models_path = Path(models_path).resolve()

        if table_path is None:
            table_path = (models_path / 'lookup_table.json').resolve()
        self.table_path = table_path
```

```python
def save_model(self, model, path=None, params=None):
    if path is None:
        path = self.model_path(params)

    path.parent.mkdir(parents=True, exist_ok=True)

    self.save_method(model, path)

    return self

def load_model(self, path):
    return self.load_method(path)

def provide_model(
    self,
    creator_method=None,
    creator_name=None,
    params=None,
    hidden_params=None,
    save: bool = False,
    load: bool = False):

    import pickle
```

FI LOSS

```python
class FScore(MeanMetricWrapper):
    def __init__(
        self,
        beta=1.0,
        name='f_score',
        dtype=None
    ):
        from functools import partial
        super(FScore, self).__init__(
            partial(custom_metrics.probabilistic.f_score, beta=beta),
            name=name,
            dtype=dtype
        )
        self.beta = beta

class F1Score(MeanMetricWrapper):
    def __init__(
        self,
        name='f1_score',
        dtype=None
    ):
        super(F1Score, self).__init__(
            custom_metrics.probabilistic.f1_score,
            name=name,
            dtype=dtype
        )
```

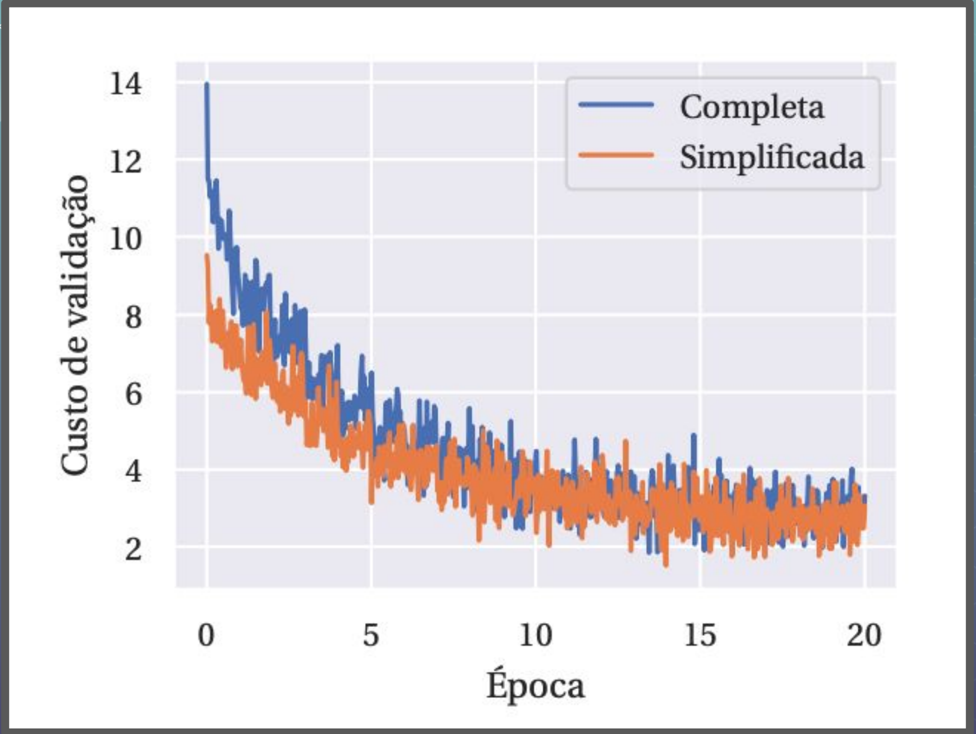
```python
class FLoss(LossFunctionWrapper):
    def __init__(
        self,
        beta=1.0,
        name='f_loss'
    ):
        from functools import partial
        super(FLoss, self).__init__(
            partial(custom_metrics.probabilistic.f_loss, beta=beta),
            name=name
        )
        self.beta = beta

class F1Loss(LossFunctionWrapper):
    def __init__(
        self,
        name='f1_loss'
    ):
        super(F1Loss, self).__init__(
            custom_metrics.probabilistic.f1_loss,
            name=name
        )
```

TREINAMENTO

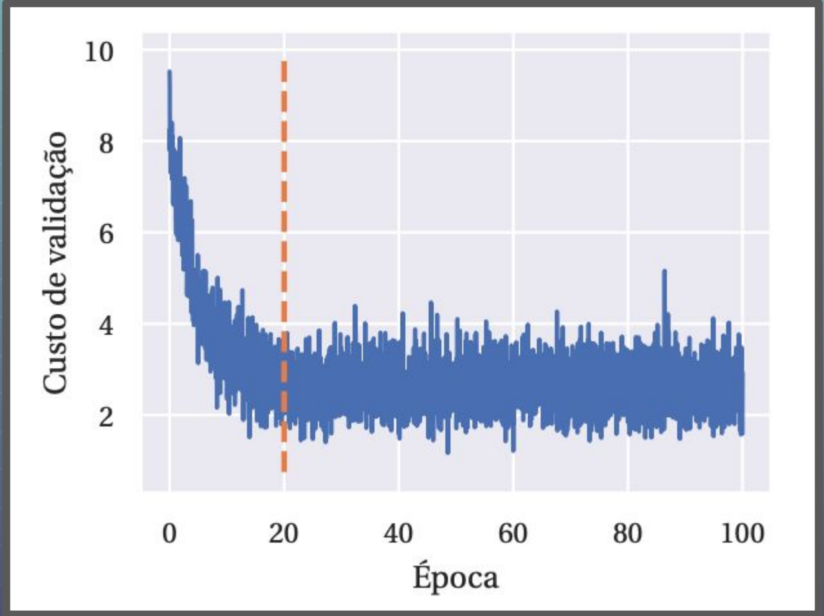
```python
    model = base_net_dict[base_net](**base_net_params)

    Optimizer = optimizer_dict[optimizer_name]
    optimizer = Optimizer(**optimizer_params)

    model.compile(
        optimizer,
        loss=loss_dict[loss],
        metrics=[metrics_dict[name] for name in metrics_names]
    )

    model.summary()

    history = model.fit_generator(
        generator=train_gen,
        validation_data=val_gen,
        class_weight=merge_dicts(
            {'none': negative_class_weight, n_labels: negative_
            {cls: positive_class_weight for cls in attr_binari
            {cls: positive_class_weight for cls in range(n_labels)}
        ),
```

```python
        ),
        train_gen=train_gen,
        val_gen=val_gen,
        callback_dict=callback_dict,
        loss_dict=loss_dict,
        callback_params={
            'ModelCheckpoint': dict(
                filepath=str(manager.models_path / 'model'),
                monitor='val_loss',
                save_best_only=True
            ),
            'ReduceLROnPlateau': dict(
                monitor='val_loss',
                factor=0.5,
                patience=5,
                verbose=1,
                mode='auto',
                min_delta=0.0001,
                cooldown=5,
                min_lr=1e-6
            ),
            'EarlyStopping': dict(
                monitor='val_loss',
```

RESULTADOS

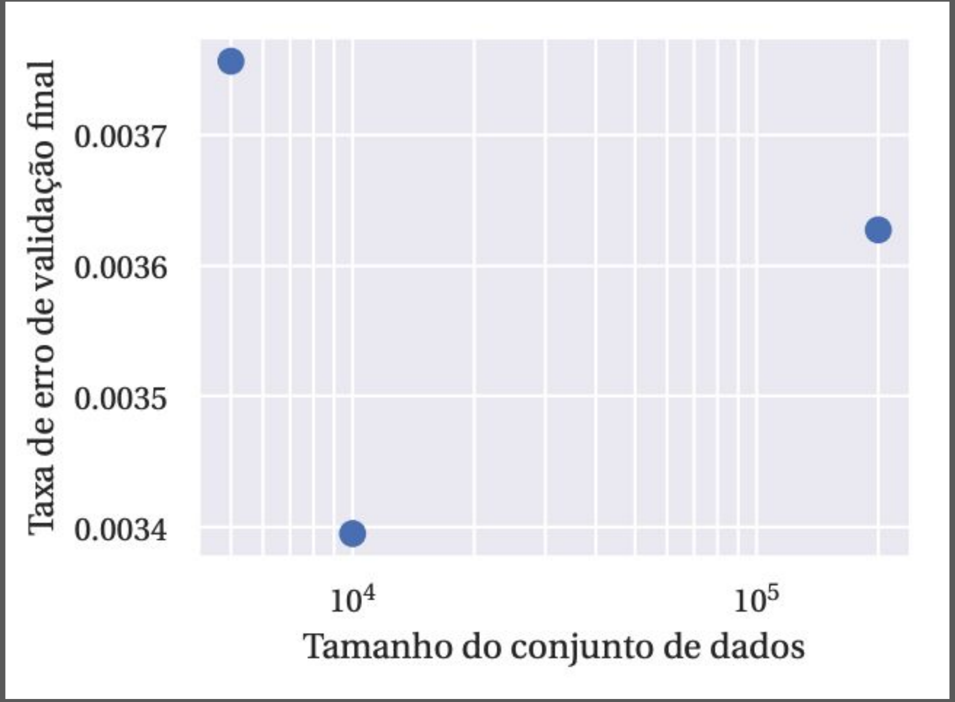


Completa
Simplificada
Custo de validação
Época
14
12
10
8
6
4
2
0
5
10
15
20

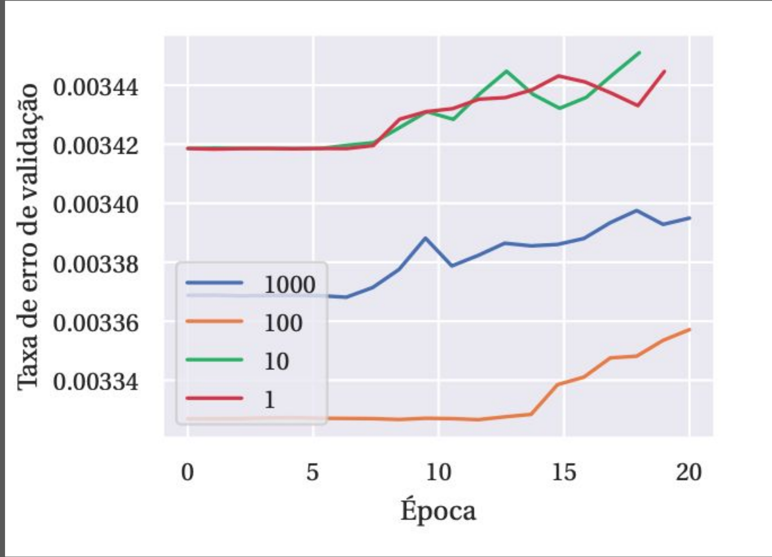


Custo de validação
Época
10
8
6
4
2
0
20
40
60
80
100

RESULTADOS

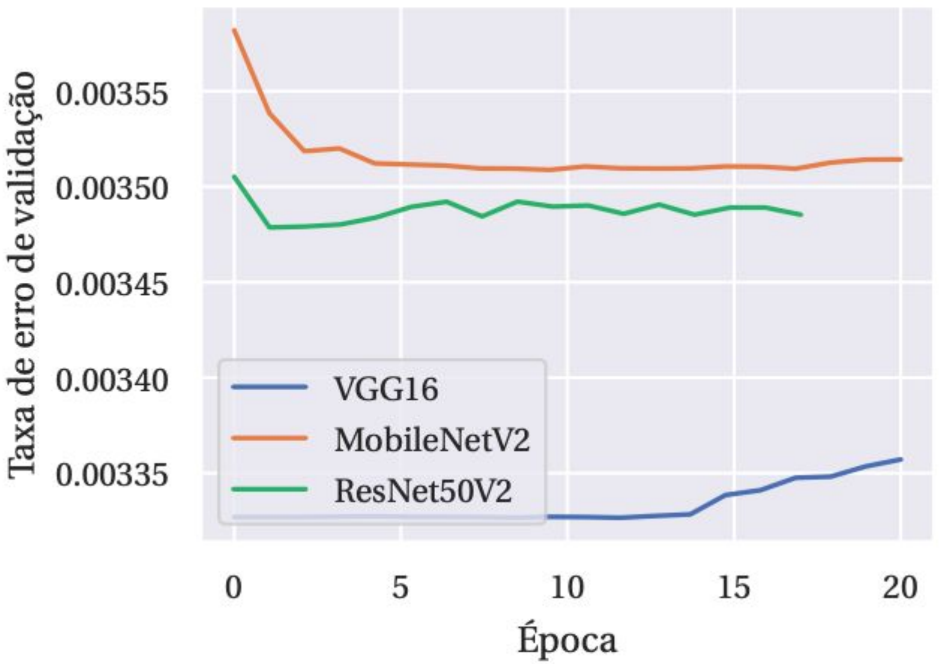


Taxa de erro de validação final
0.0037
0.0036
0.0035
0.0034
10^4
10^5
Tamanho do conjunto de dados



Taxa de erro de validação
0.00344
0.00342
0.00340
0.00338
0.00336
0.00334
1000
100
10
1
0
5
10
15
20
Época

RESULTADOS



Modelo	Taxa de Erro
exp1	0.00756
proj1	0.01912
proj2	0.00731
proj3	0.00956
proj4	0.00901
proj5	0.00825

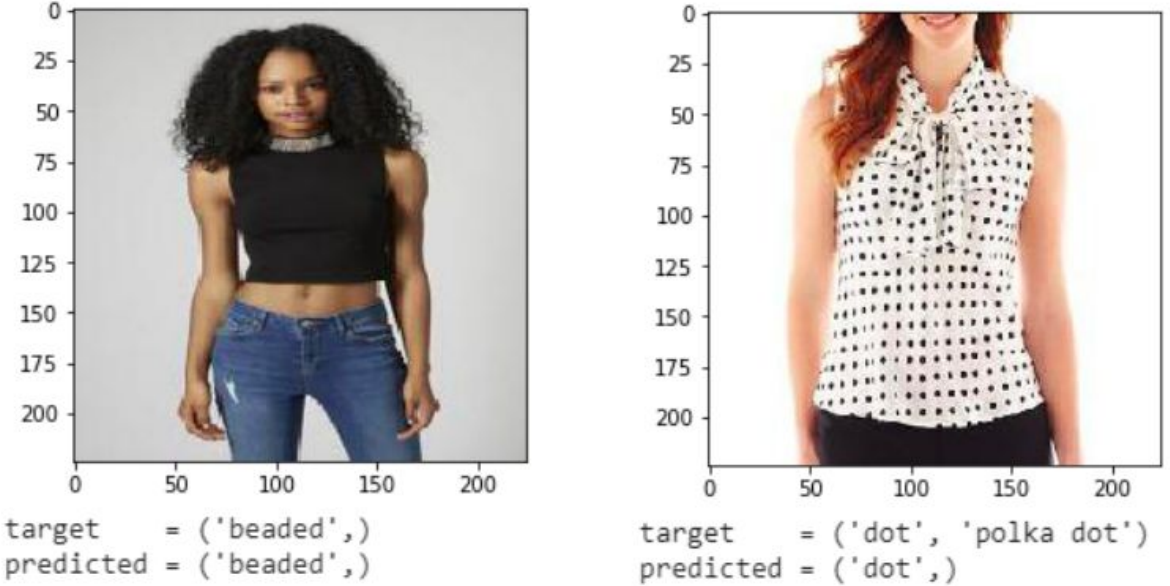
Fonte: dos autores.

RESULTADOS

Modelo	Taxa de Erro
exp1	0.00756
proj1	0.01912
proj2	0.00731
proj3	0.00956
proj4	0.00901
proj5	0.00825

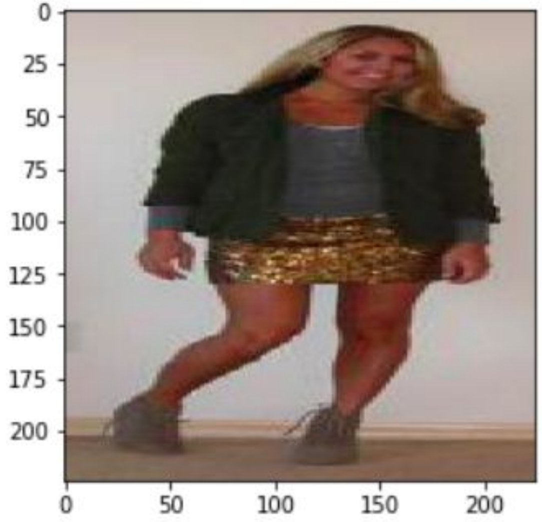
Nome	Máquina	N	Rede *Landmark*	w_N	Épocas	Batch Size	Workers
exp1	nvidia2	50,000	Relevada	100	100	32	16
proj1	cpu1	10,000	Presente	10	20	16	4
proj2	nvidia1	10,000	Relevada	1000	20	32	16
proj3	nvidia2	10,000	Presente	100	20	32	10
proj4	nvidia1	20,000	Presente	100	20	40	14
proj5	nvidia1	50,000	Presente	200	20	50	10

PREDIÇÕES



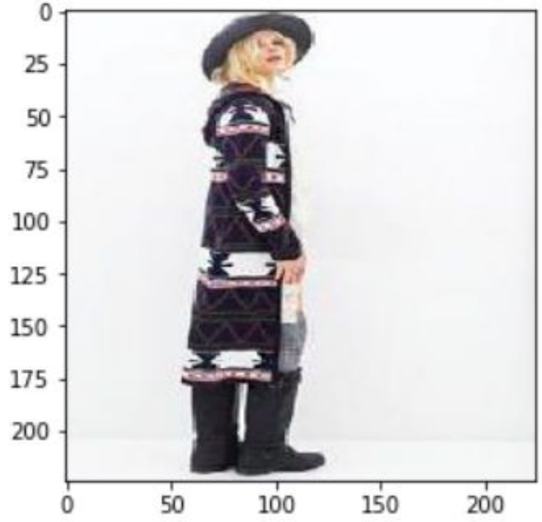
(a) Predição correta. **(b)** Predição parcialmente correta.

PREDIÇÕES




```
target    = ('everyday', 'lace-up')
predicted = ()
```

(c) Predição vazia.




```
target    = ('camo', 'hood', 'utility', 'zipper')
predicted = ('classic',)
```

(d) Predição errônea.

CONCLUSÕES

A

Architecture

VGG16.

W

Loss Weight

10^-2.

E

Epochs

20

L

Landmarks

Só introduzem erros neste N.

OBRIGADO

slidesgo