Animal Classifier

Marko Bukovina a Martin Kvietok

EDA Analysis

DATASET PREVIEW

cane
cane/OIP-rA-sxQehLnbtrlasmqRsagHaFj.jpeg
300×225px
brightness=96.9
.jpeg, RGB, 8-bit

cavallo
cavallo/OIP-GtjY8Gnm_4cZwBzbtnP2eAHaE8.jpeg
300×200px
brightness=120.3
.jpeg, RGB, 8-bit

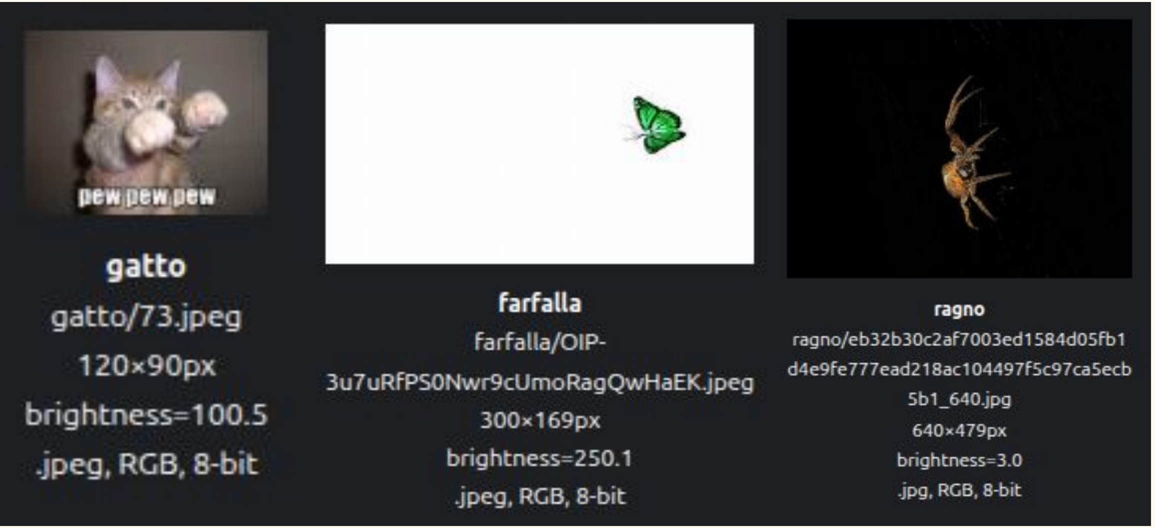
farfalla
farfalla/OIP-HnssZSKncsXlfiJOgvCWQAHaFi.jpeg
300×225px
brightness=140.1
.jpeg, RGB, 8-bit

gatto
gatto/ec37b10721f01c22d2524518b7444f92e37fe5d404b0144390f8c079a4e5b6_640.jpg
640×469px
brightness=160.0
.jpg, RGB, 8-bit

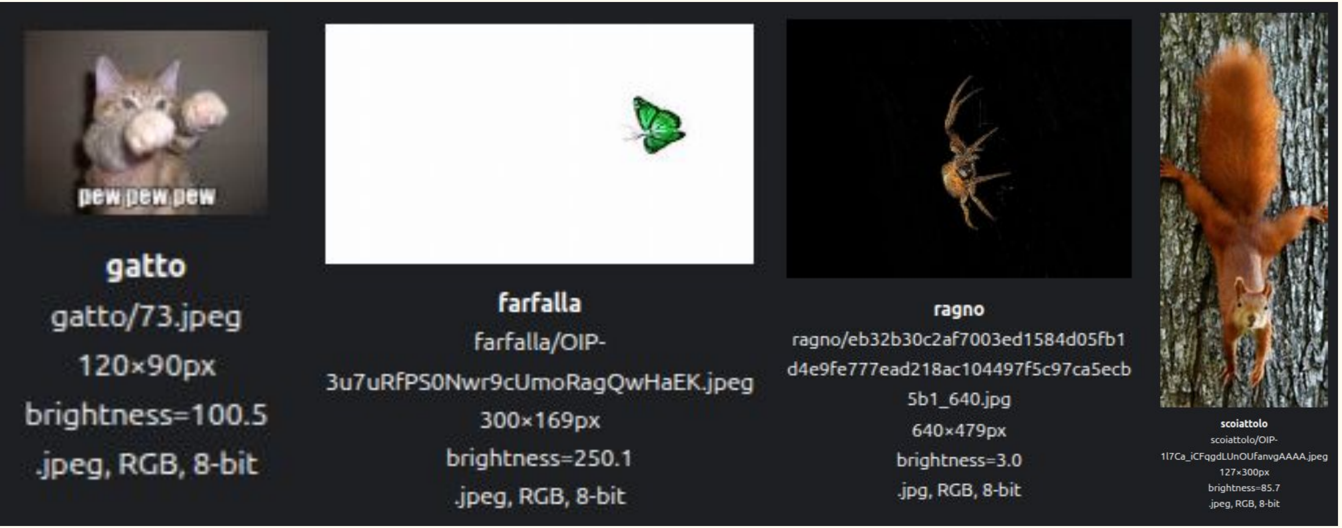
```
['cane', 'cavallo', 'elefante', 'farfalla', 'gallina', 'gatto', 'mucca', 'pecora', 'ragno', 'scoiattolo']
```

DATASET OUTLIERS

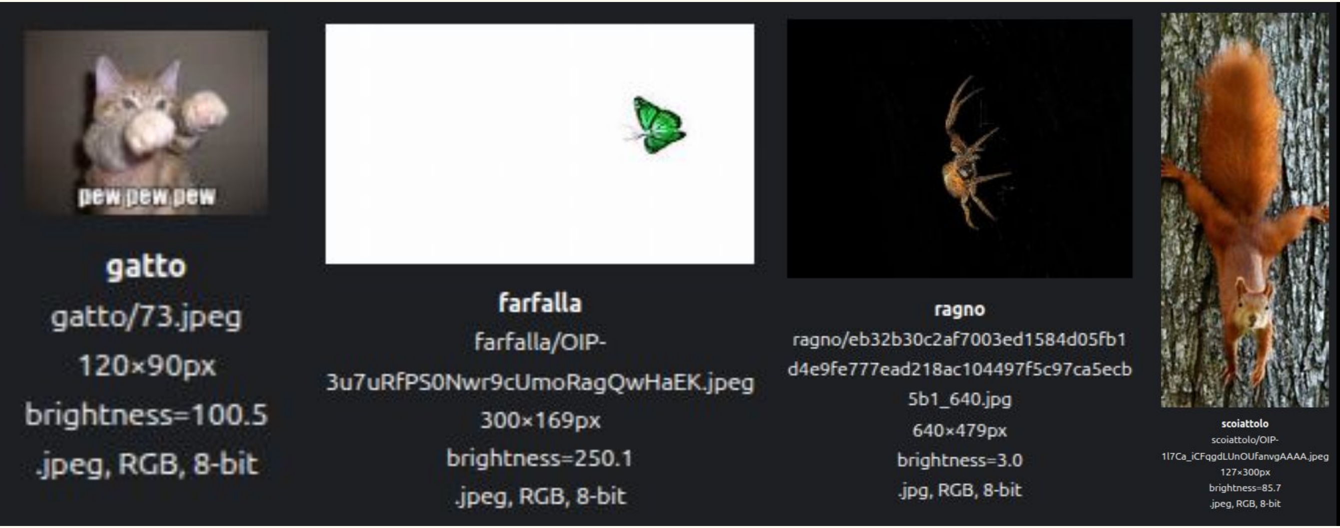
DATASET OUTLIERS



DATASET OUTLIERS



DATASET OUTLIERS



DATASET INFO

Positives

- Large number of samples **(24k)** and an appropriate number of classes **(10)**
- No **corrupted or unreadable images**
- Consistent color models (RGB)
- Correct image formats (.jpeg)
- High data quality for classification tasks
- **Image brightness is not an issue** (outliers are kept in the dataset)

Identified Issues

- Significant **class imbalance**
- 2,672 images with **atypical dimensions** (outliers)
- Minor presence of **incorrectly labeled** samples

Preprocessing

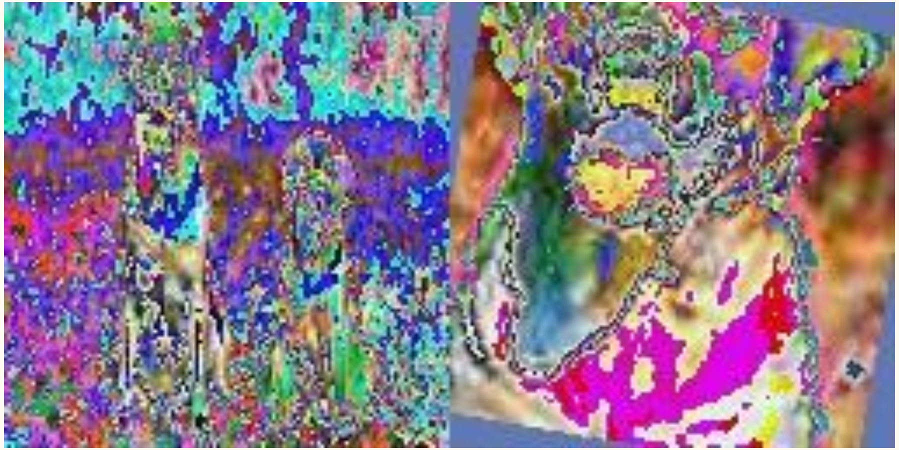
DATA PROCESSING

DATA SPLIT

- 90/10
- BATCHSIZE 64

PREPROCESSING

- 128x128px
- pixel scaling [0,1]
- norm by **mean,std**



DATA AUGMENTATION

- **RandomResizedCrop(target_size)**
- **RandomHorizontalFlip()**
- **RandomRotation(10)**

We also used **minority-augmentation boosting** to balance the dataset (applying stronger augmentation to underrepresented classes)

Model

Configuration

```
EPOCHS = 100

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(img_class_model.parameters(), lr=0.2, momentum=0.9)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, "max", patience=4)
e_stop = EarlyStopping(patience=5, diff=0.01)
```

Configuration

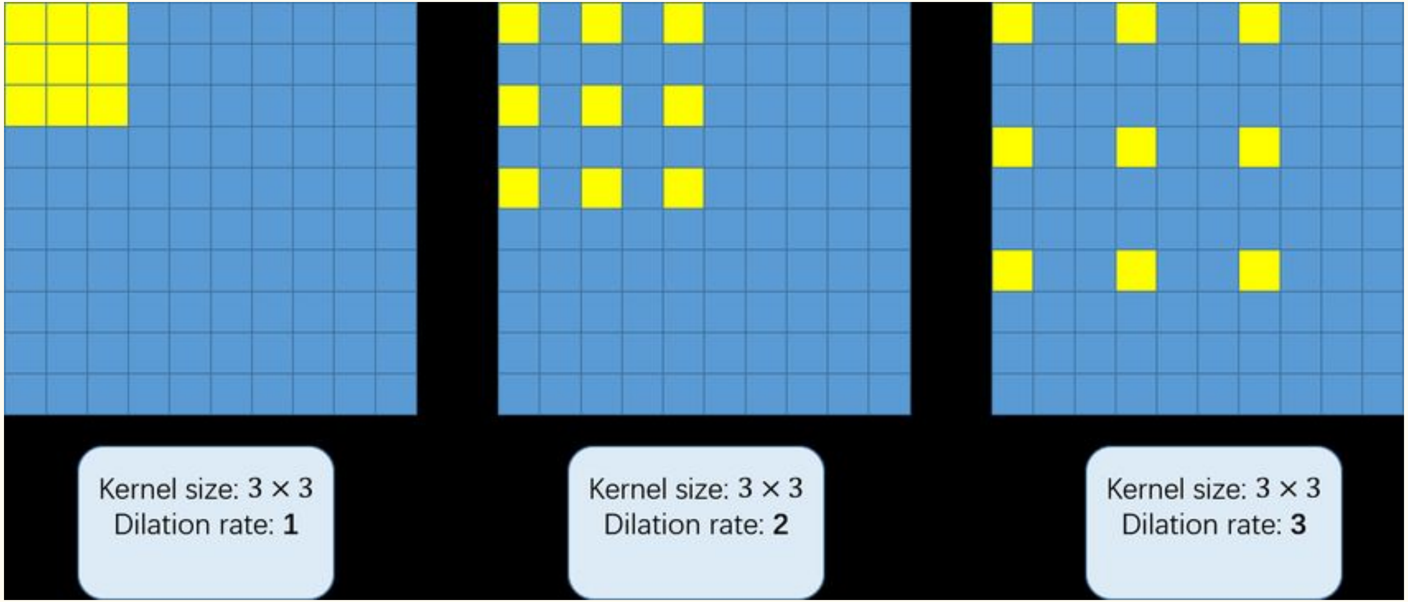
```
EPOCHS = 100

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(img_class_model.parameters(), lr=0.2, momentum=0.9)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, "max", patience=4)
e_stop = EarlyStopping(patience=5, diff=0.01)
```

Model Architecture

```
# Block 1: 256 -> 128
self.block1 = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(32, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.LeakyReLU(inplace=True),
    nn.MaxPool2d(2)
)
```

```python
# Block 2: 128 -> 64
self.block2 = nn.Sequential(
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, dilation=2, padding=2),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(inplace=True),
    nn.MaxPool2d(2)
)
```

```python
# Block 2: 128 -> 64
self.block2 = nn.Sequential(
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, dilation=2, padding=2),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(inplace=True),
    nn.MaxPool2d(2)
)
```


Kernel size: 3 × 3
Dilation rate: 1
Kernel size: 3 × 3
Dilation rate: 2
Kernel size: 3 × 3
Dilation rate: 3

```python
# Block 3: 64 -> 32
self.block3 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, dilation=2, padding=2),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, dilation=3, padding=3),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(inplace=True),
    nn.MaxPool2d(2)
)
```

```python
# Block 4: 32 -> 16 (Grad-CAM)
self.block4 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, dilation=2, padding=2),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(inplace=True),
    nn.MaxPool2d(2)
)
```

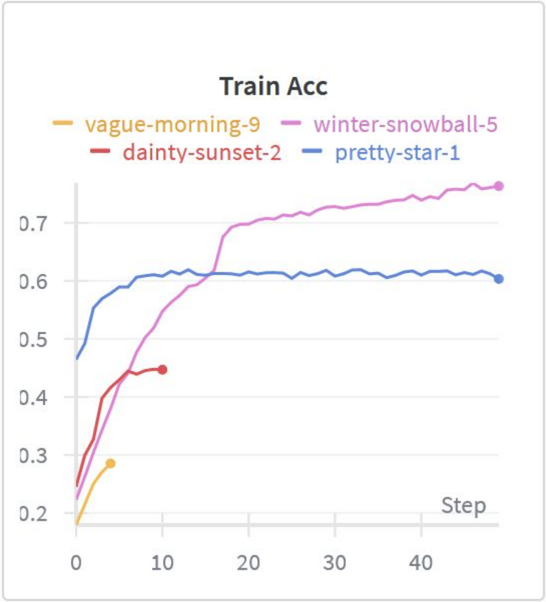
```python
# Global average pooling reduces (256×16×16) to (256)
self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))

# Classification head
self.fc1 = nn.Linear(256, 128)
self.fc2 = nn.Linear(128, self.numberOfClasses)
```

Training

Beginnings

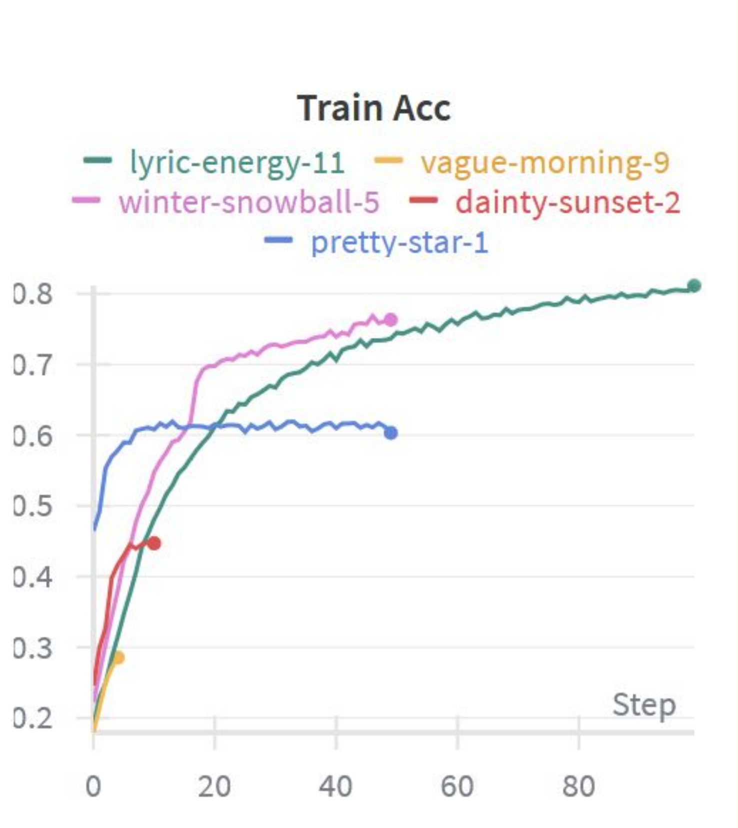
Image resolution 256x256

```
self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1,padding=1)
self.bn1 = nn.BatchNorm2d(16)

self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_si
self.bn2 = nn.BatchNorm2d(32)

self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_si
self.bn3 = nn.BatchNorm2d(64)

self.fc1 = nn.Linear(in_features=64*32*32, out_features=128)
self.fc2 = nn.Linear(in_features=128, out_features=self.numberOfC
```

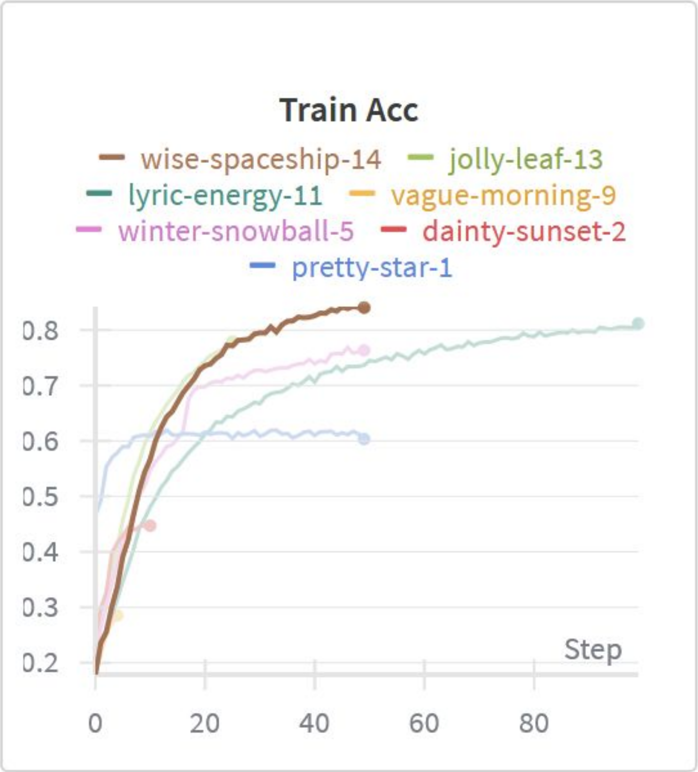


```python
    # Block 4: 32 -> 16 (Grad-CAM)
    self.block4 = nn.Sequential(
        nn.Conv2d(128, 256, kernel_size=3, padding=1),
        nn.BatchNorm2d(256),
        nn.LeakyReLU(inplace=True),
        nn.MaxPool2d(2)
    )
    # Global average pooling reduces (256×16×16) to (256)
    self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))

    # Classification head
    self.fc1 = nn.Linear(256, 128)
    self.fc2 = nn.Linear(128, self.numberOfClasses)

def forward(self, x):
    x = self.block1(x)
    x = self.block2(x)
    x = self.block3(x)
    x = self.block4(x)
    x = self.global_avg_pool(x)
    x = torch.flatten(x, 1)
    x = fun.leaky_relu(self.fc1(x))
    x = self.dropout(x)
    return self.fc2(x)
```

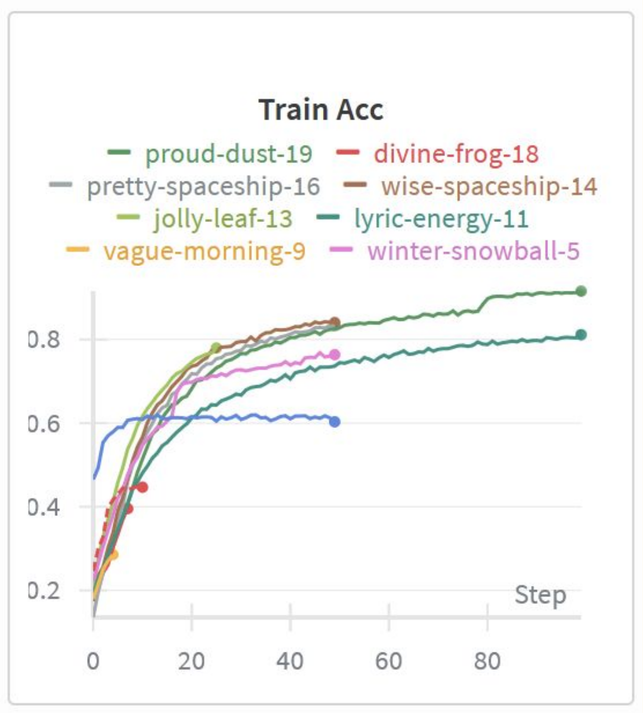

Pred: scoiattolo
151930 fotosearch ©

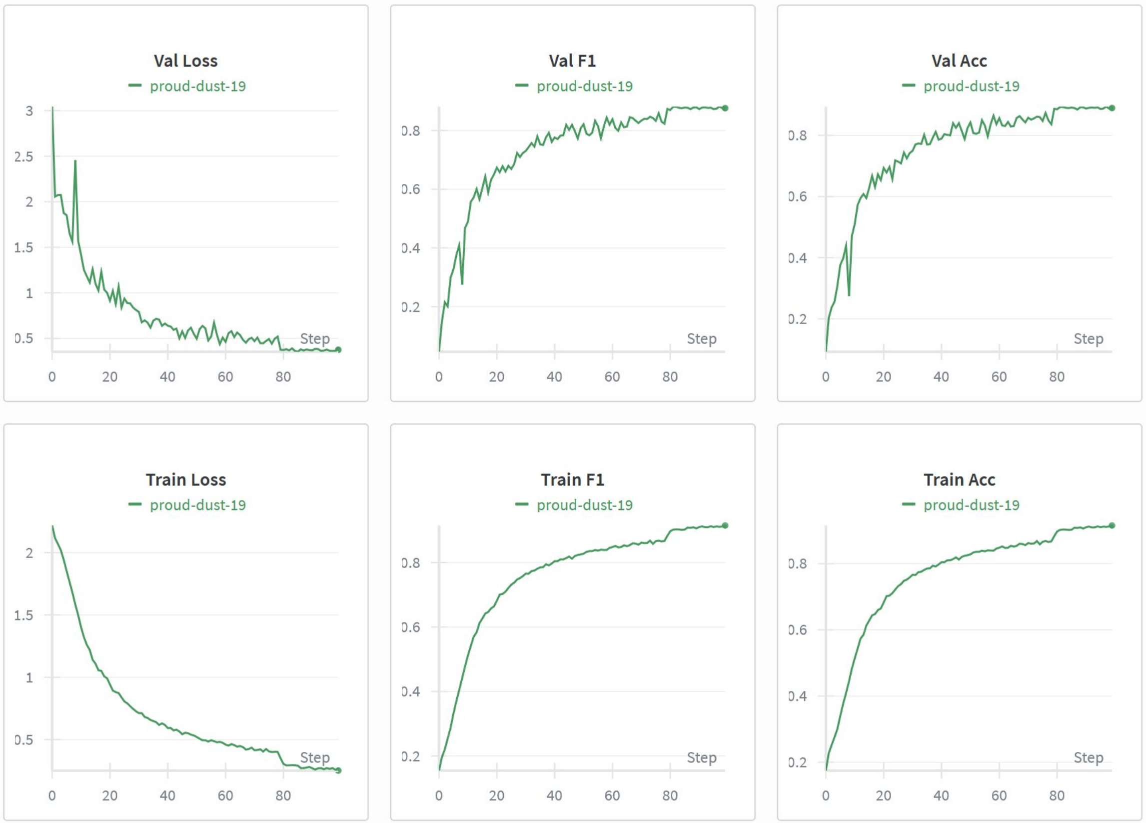

Train Acc
lyric-energy-11
vague-morning-9
winter-snowball-5
dainty-sunset-2
pretty-star-1
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Step
0
20
40
60
80

More convolution layers



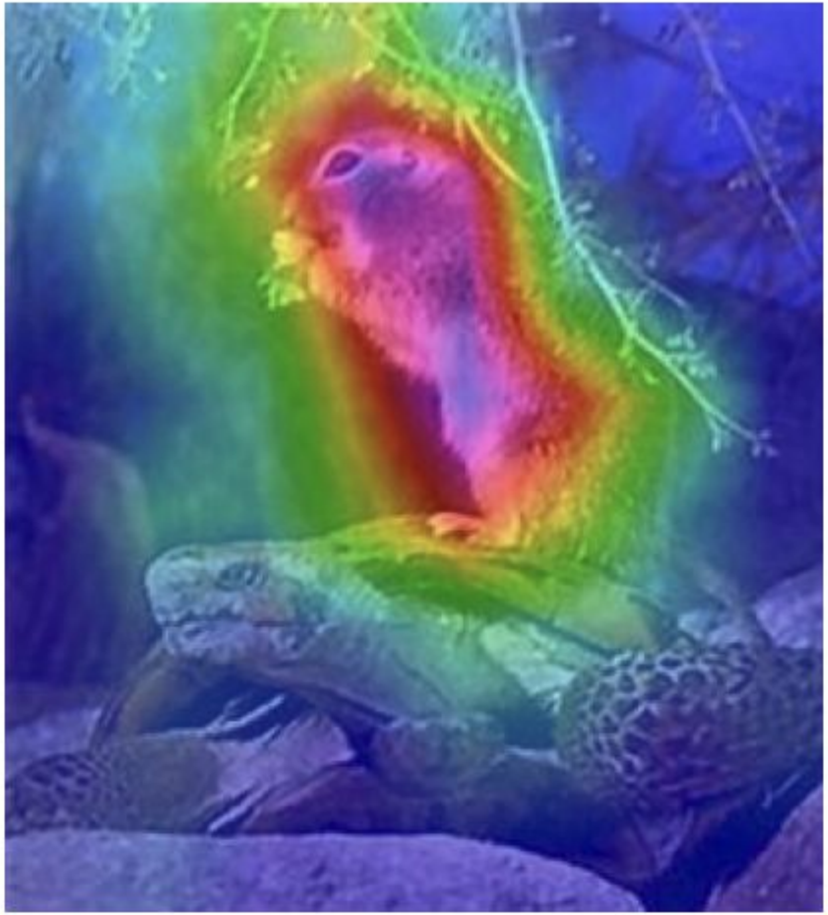
Final adjustments



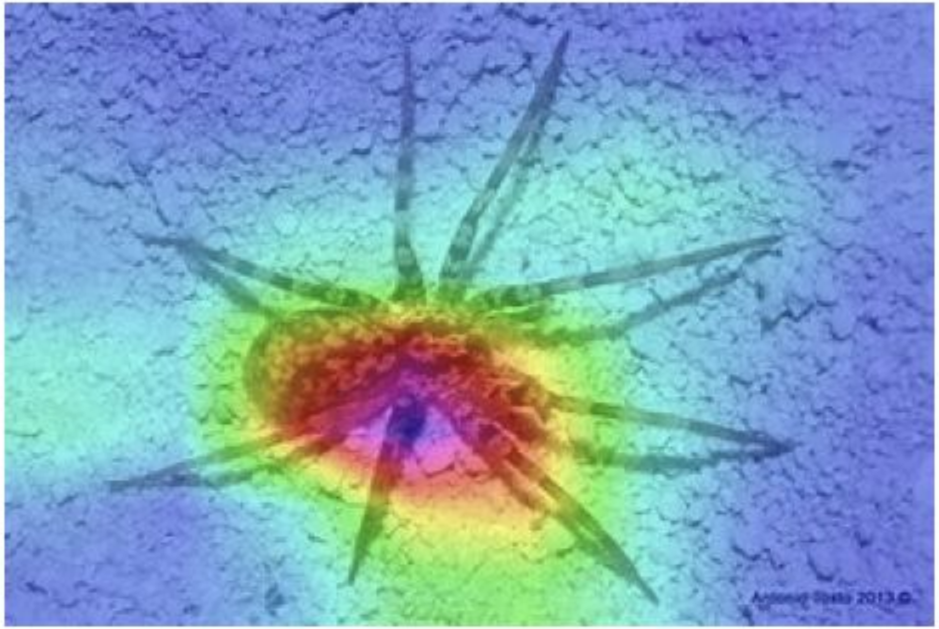


Pred: mucca

Pred: scoiattolo

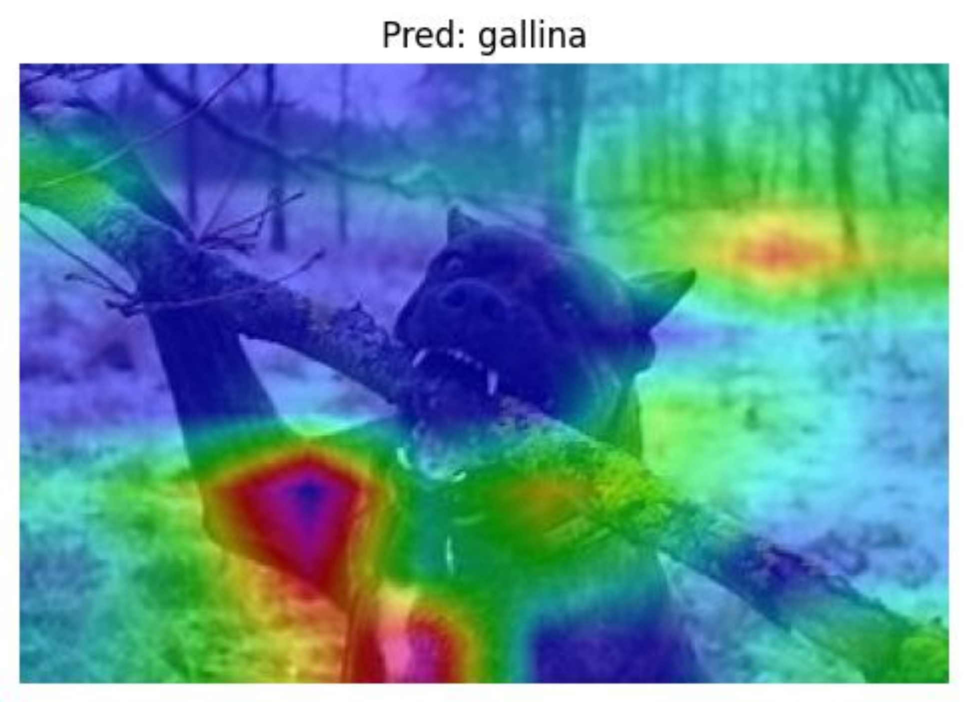


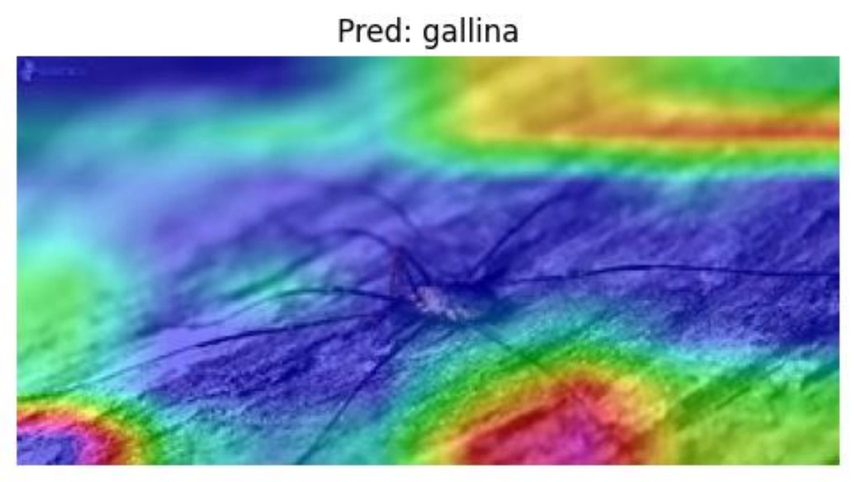
Pred: ragno



Pred: pecora

Mistakes





Results

KEY FINDINGS

- **The model successfully learned to classify animal images with high accuracy (87.6% on the test set)**
- Overfitting max 2-3%