TRANSFER LEARNING

IDEA

- Deep Neural Networks wymagają dużych zbiorów danych:
  1. nie zawsze są dostępne
  2. pozyskanie może być kosztowne
  3. trenowanie zajmuje dużo czasu

- Czy można wykorzystać model wytrenowany w jednej dziedzinie do klasyfikacji w innej?

Nasz mózg tak działa.

1. Łatwiej jest nauczyć się jeździć na rowerze, gdy się umie jeździć na hulajnodze.
2. Umiejąc grać na jednym instrumencie, łatwiej jest opanować inny, szczególnie podobny.

IDEA

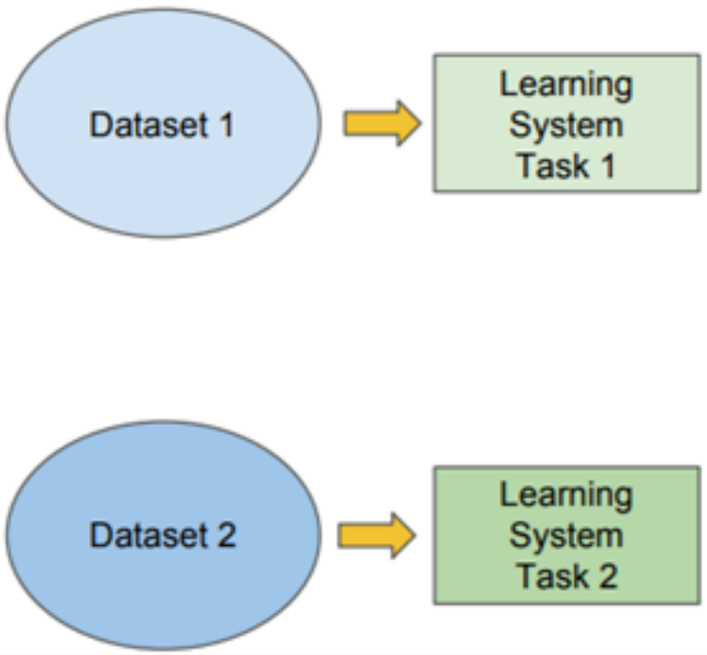
- Algorytmy ML są zazwyczaj wyspecjalizowane do jednego konkretnego zadania
- Transfer learning: wykorzystanie wiedzy uzyskanej przy modelowaniu jednego zadania w innym, podobnym.

AFTER SUPERVISED LEARNING – TRANSFER LEARNING WILL BE THE NEXT DRIVER OF ML COMMERCIAL SUCCESS

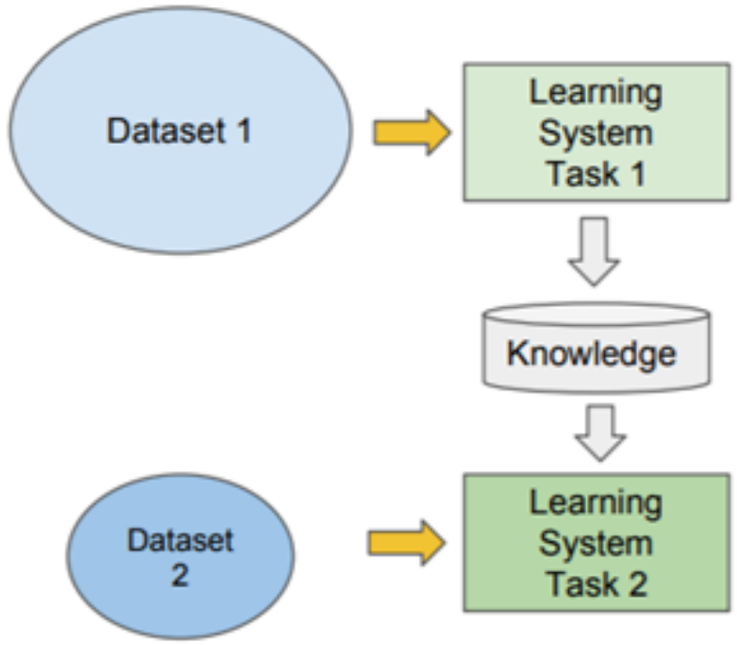
– Andrew Ng

Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



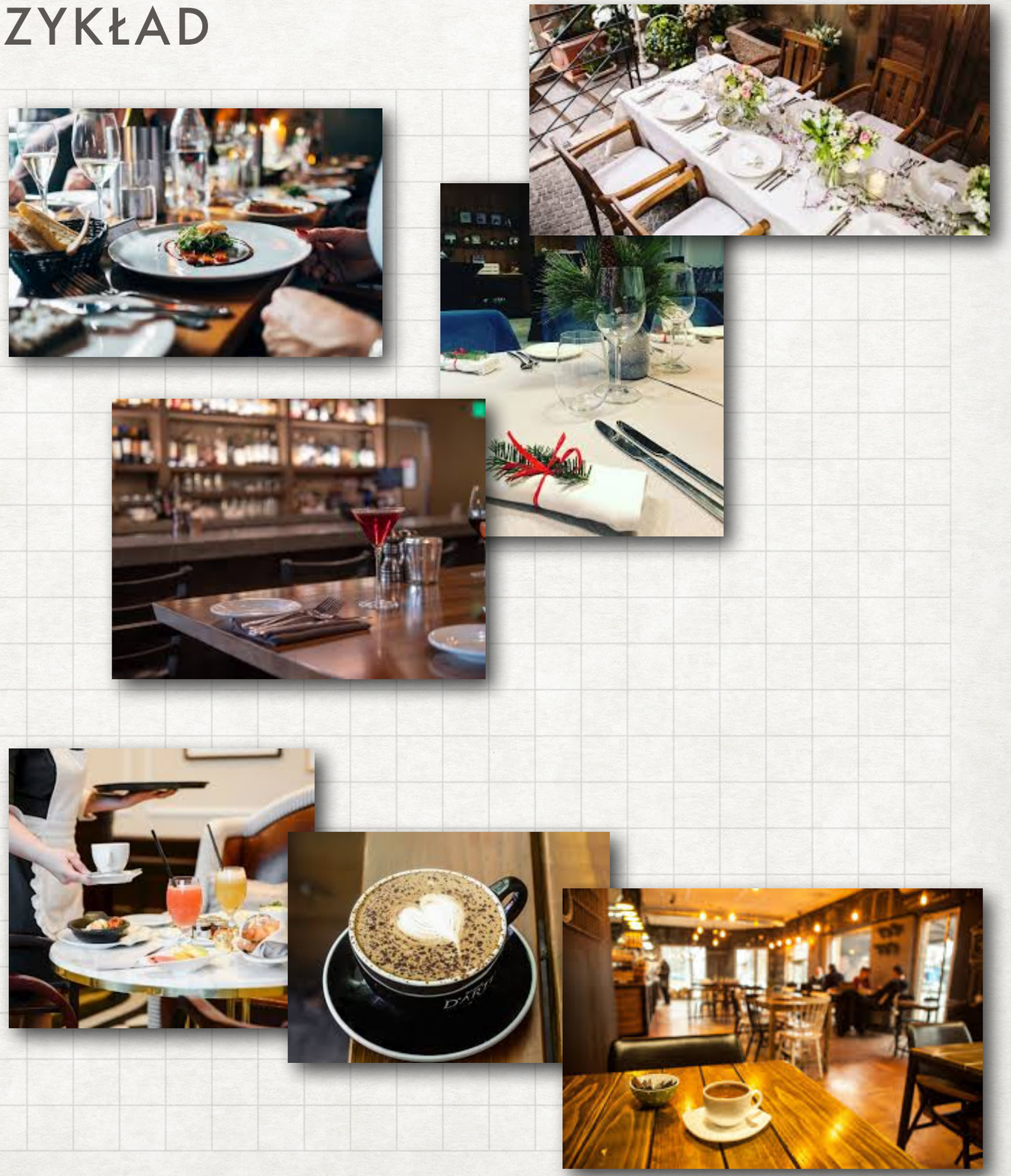
- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



OGÓLNY ZARYS

PRZYKŁAD

- Trenujemy model A rozpoznający na zdjęciach obiekty z restauracji
- Chcielibyśmy następnie rozpoznawać obiekty z kawiarni… ale mamy znacznie mniej danych.
- Bezpośrednie wykorzystanie modelu za pewne nie będzie najskuteczniejsze…
- …ale są pewne wspólne cechy obrazów występujących w obu domenach, które pozwalają nam mieć nadzieję, że da się wykorzystać model A.



DOWN THE RABBIT HOLE

DEFINICJE

A **Domain** consists of two components: $D = \{\mathcal{X}, P(X)\}$

- Feature space: $\mathcal{X}$
- Marginal distribution: $P(X)$, $X = \{x_1, \cdots, x_n\}, x_i \in \mathcal{X}$

For a given domain **D**, a **Task** is defined by two components:

$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta\} \qquad Y = \{y_1, \cdots, y_n\}, y_i \in \mathcal{Y}$$

- A label space: $\mathcal{Y}$
- A predictive function η, learned from *feature vector/label* pairs, (x_i, y_i), $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
- For each feature vector in the domain, η predicts its corresponding label: $\eta(x_i) = y_i$

DEFINICJE

A **Domain** consists of two components: $D = \{\mathcal{X}, P(X)\}$

- Feature space: $\mathcal{X}$
- Marginal distribution: $P(X)$, $X = \{x_1, \cdots, x_n\}, x_i \in \mathcal{X}$

For a given domain **D**, a **Task** is defined by two components:

$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta\} \qquad Y = \{y_1, \cdots, y_n\}, y_i \in \mathcal{Y}$$

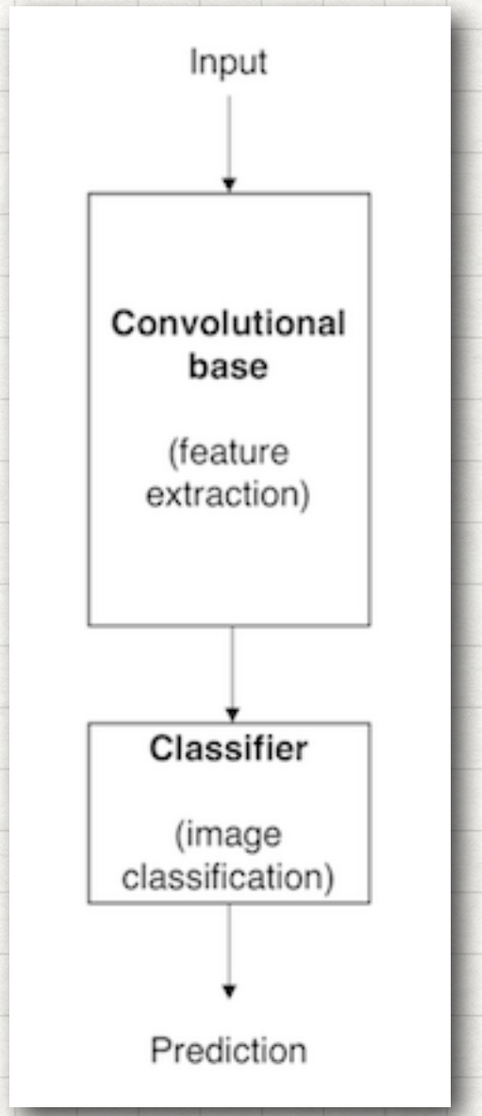
- A label space: $\mathcal{Y}$
- A predictive function η, learned from *feature vector/label* pairs, (x_i, y_i), $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
- For each feature vector in the domain, η predicts its corresponding label: $\eta(x_i) = y_i$

Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in $\mathcal{D}_T$ with the information gained from $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.

CO TO OZNACZA

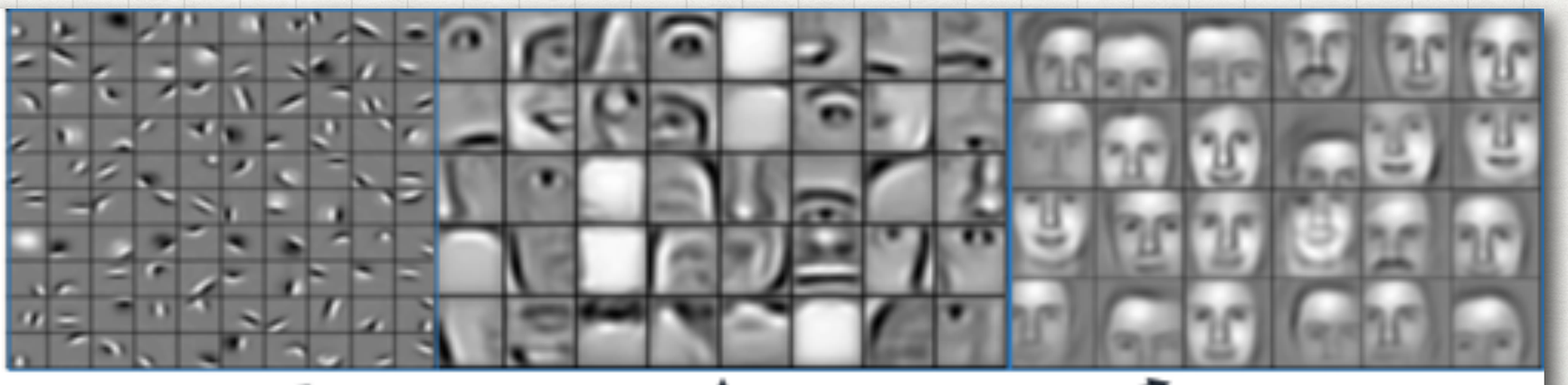
PRZYKŁAD CNN

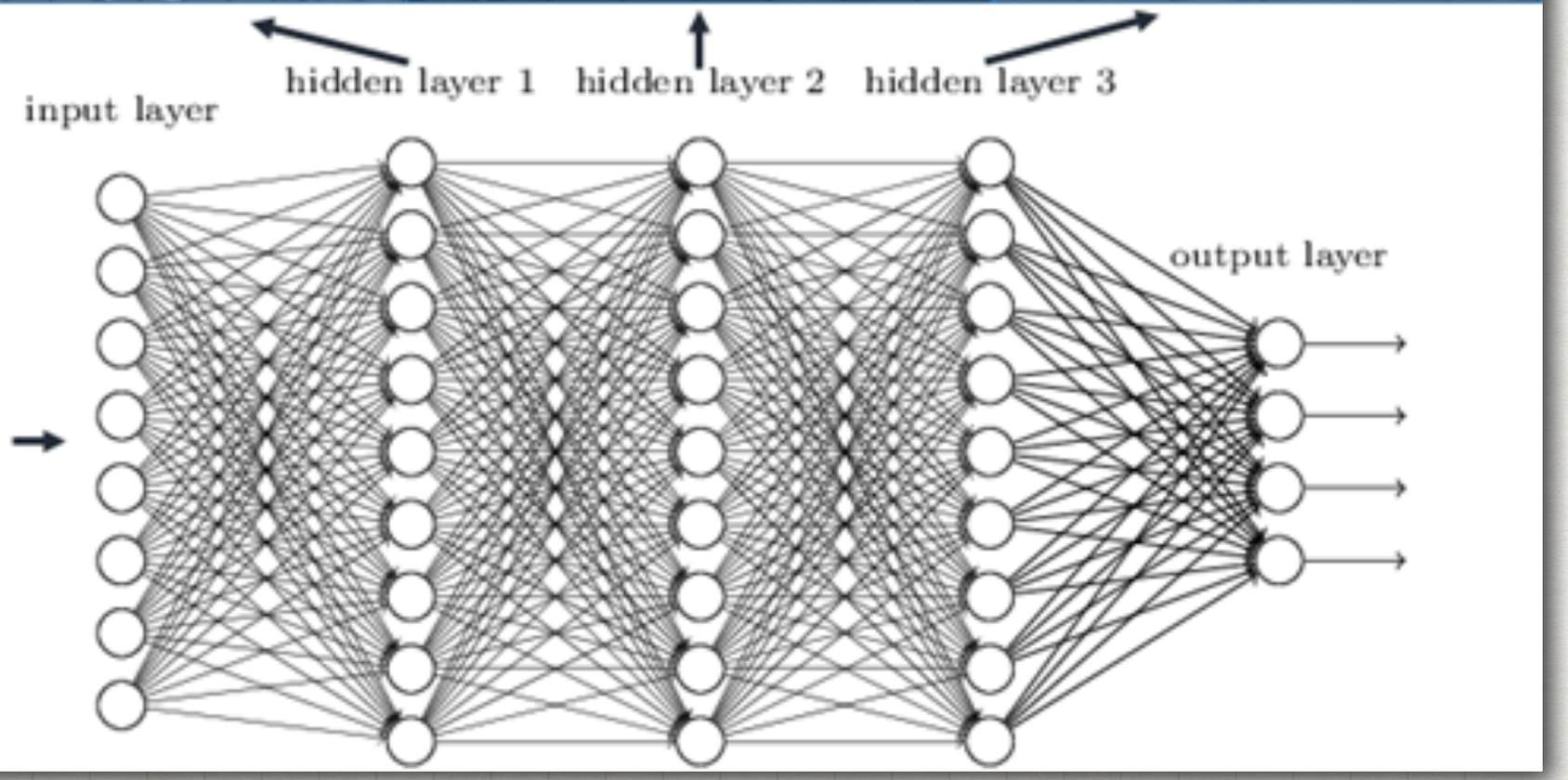
- Typową sieć konwolucyjną możemy podzielić na dwie części:
  1. feature extraction
  2. classification
- Płytkie warstwy: cechy ogólne
- Głębokie warstwy: cechy bardziej szczegółowe dla konkretnego problemu



CECHY W ZALEŻNOŚCI OD GŁĘBOKOŚCI

Deep neural networks learn hierarchical feature representations



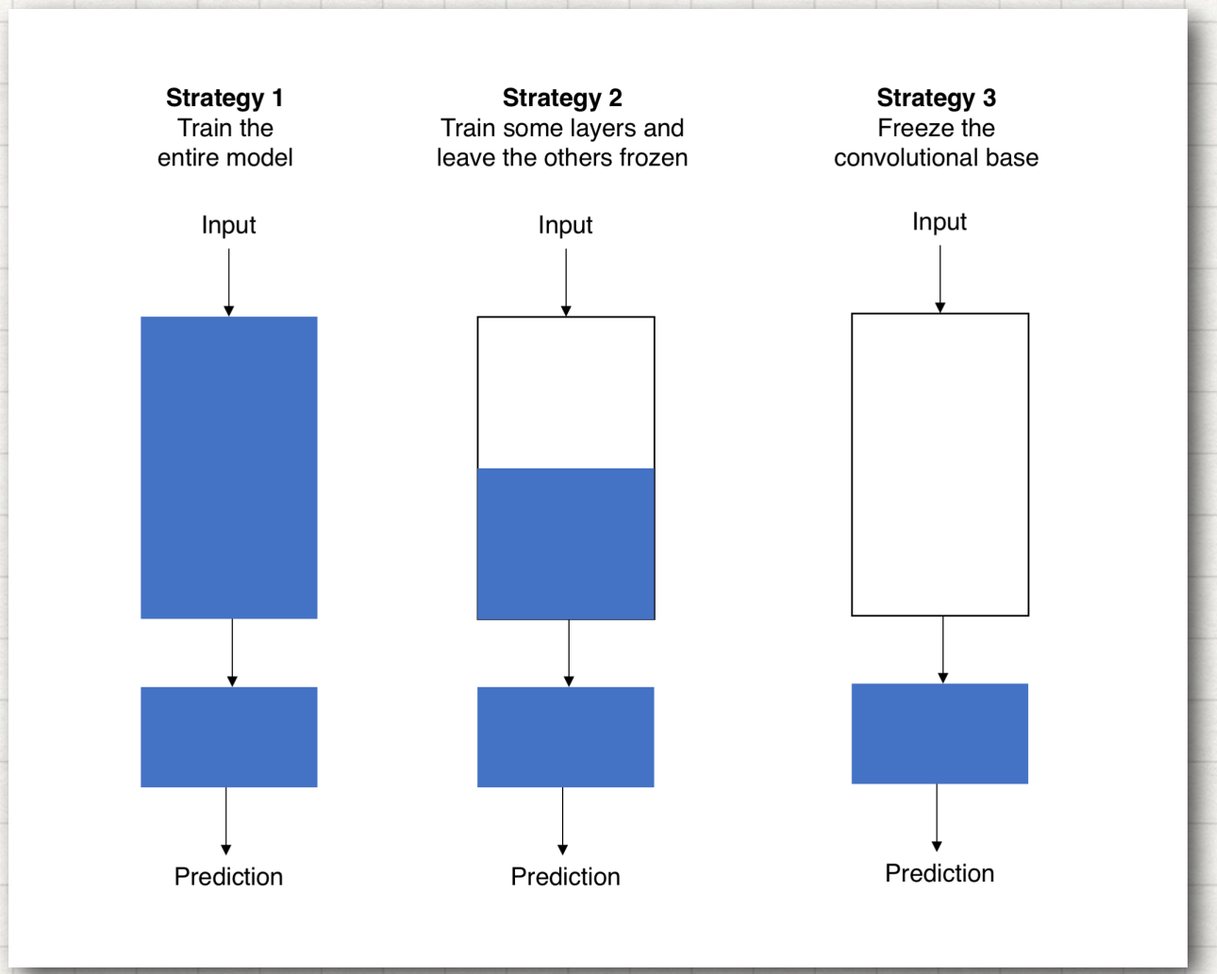


TRZY STRATEGIE

Wartości początkowe wag są w wcześniej wytrenowanego modelu.

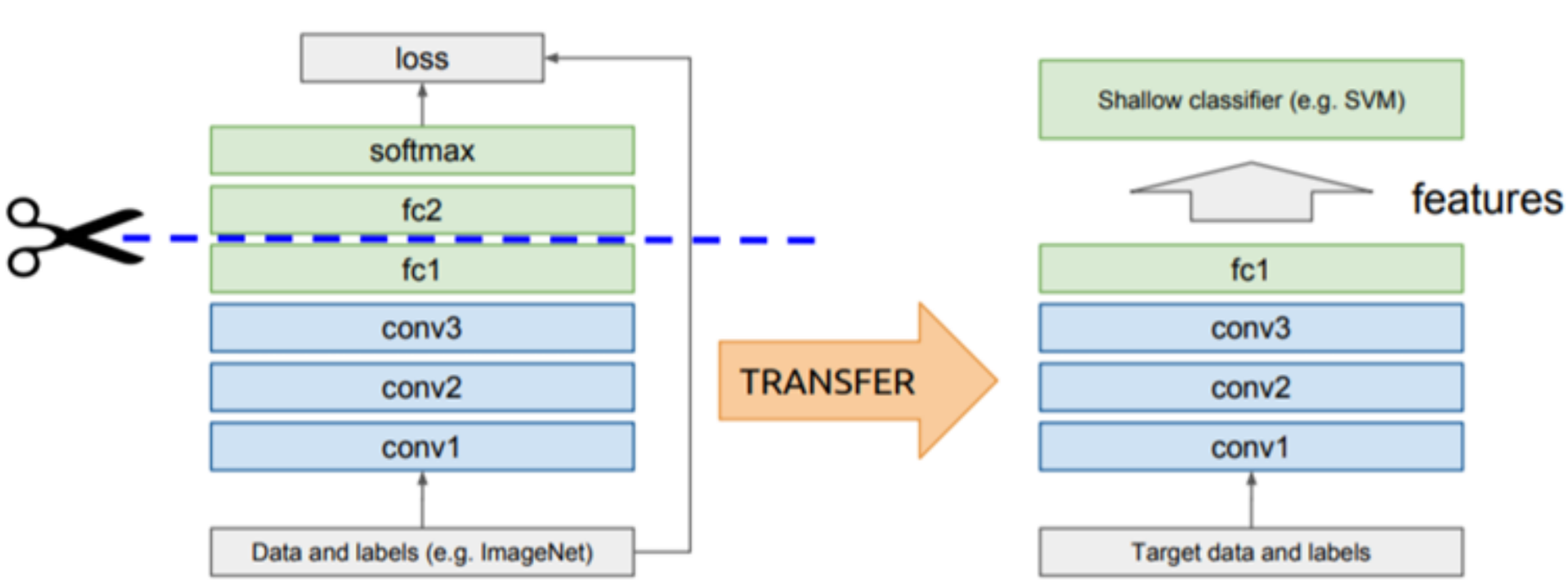
1. Trenowanie całego modelu.
2. Trenowanie kilku głębszych warstw modelu i klasyfikatora.
3. Trenowanie samego klasyfikatora.

W przypadkach 1 i 2 należy odpowiednio zmniejszyć learning rate!

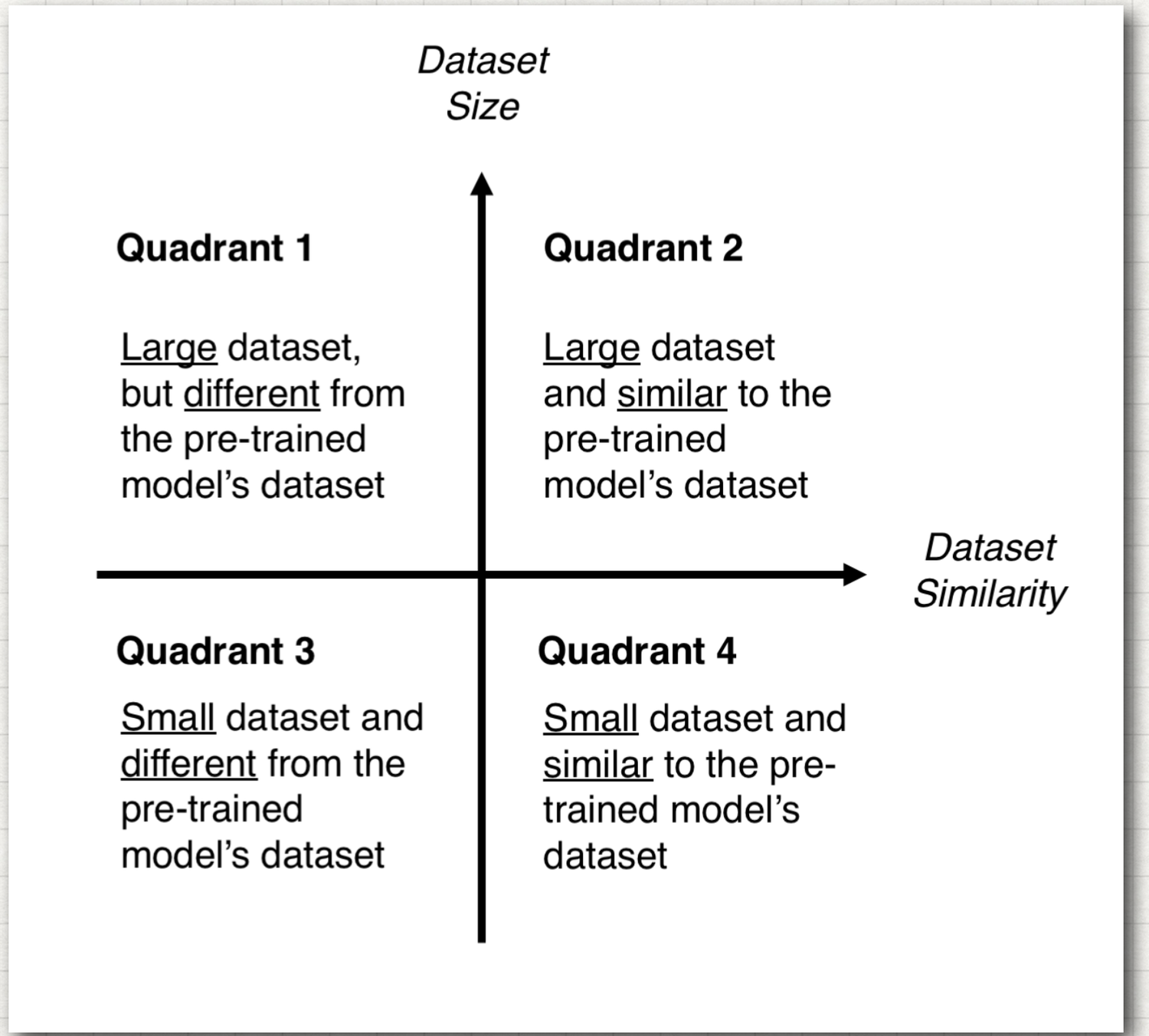


Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

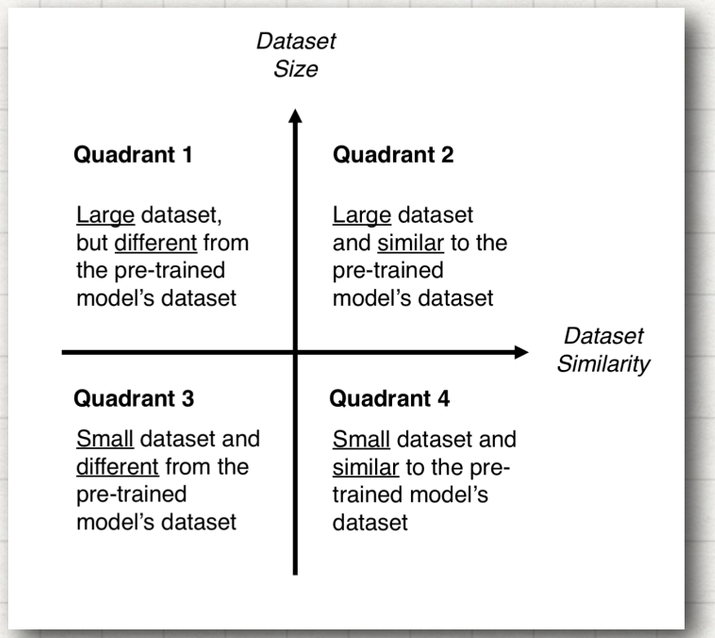
Assumes that $D_S = D_T$

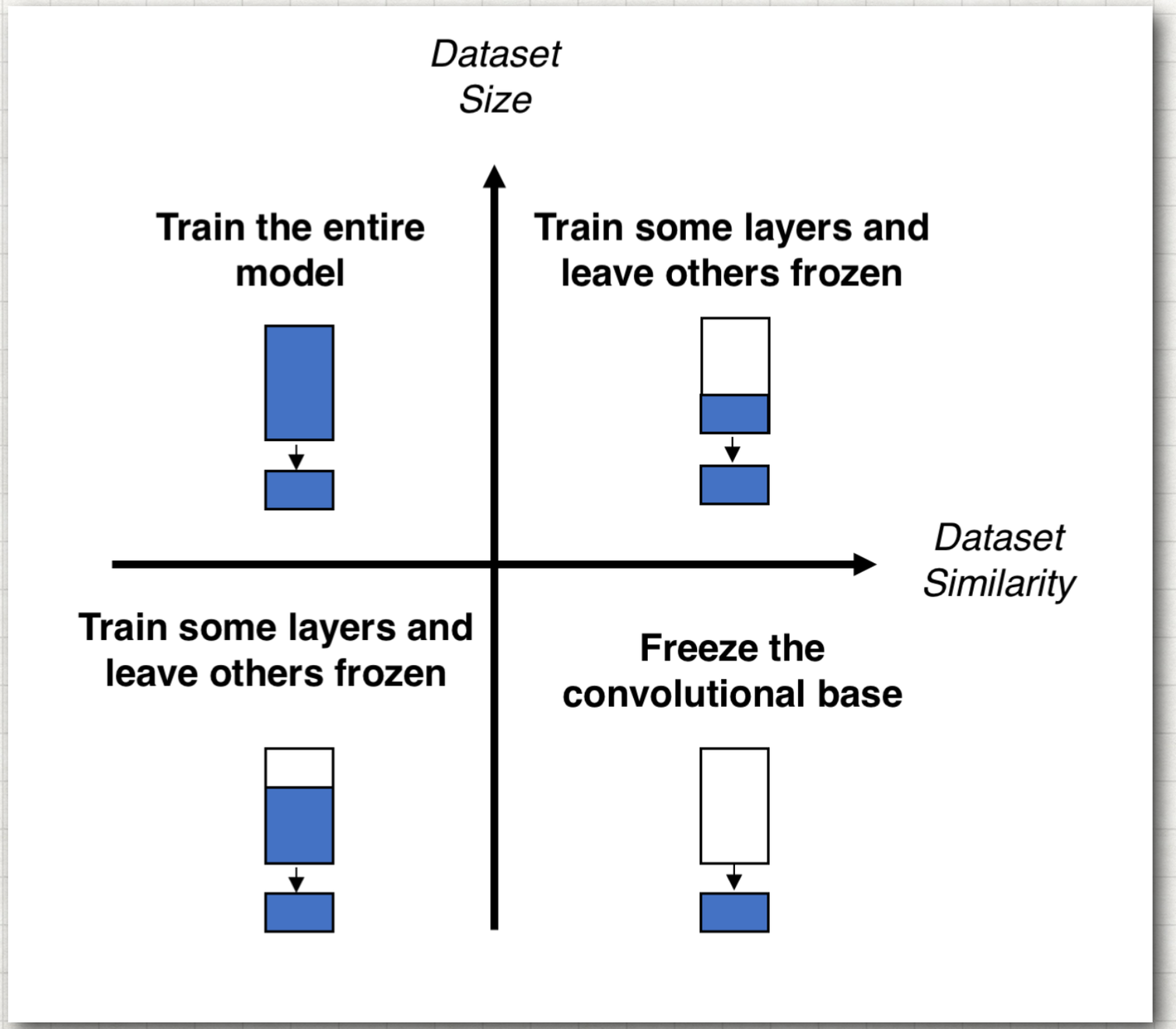


KIEDY CO STOSOWAĆ?



KIEDY CO STOSOWAĆ?





RZECZY DO ZAPAMIĘTANIA

- Co transferować

  Co jest wspólnego między domenami, zadaniami.

- Kiedy transferować

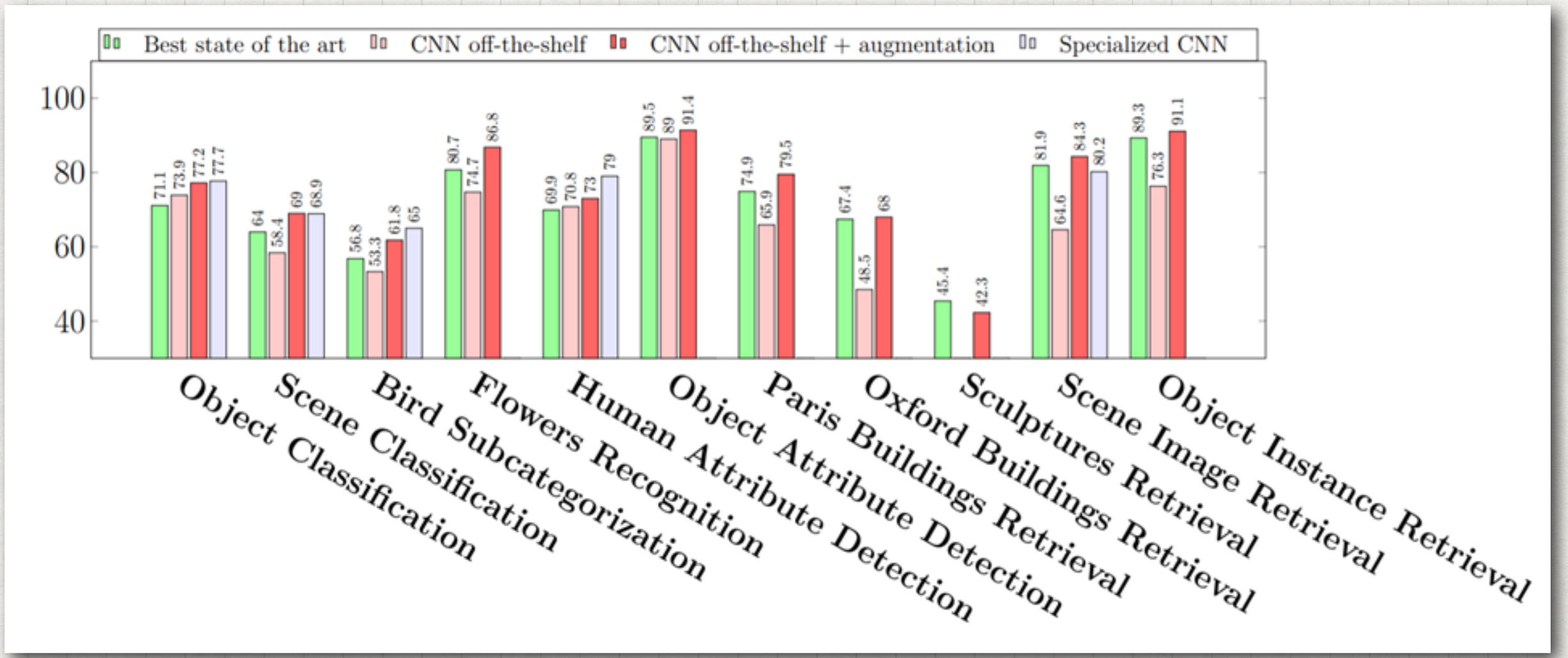
  Są sytuacje, w których transfer learning może prowadzić do gorszych rezultatów niż wyspecjalizowane zadania.

- Jak transferować

  Różne strategie, specyficzne modyfikacje, itp.

JAK SKUTECZNE JEST TRANSFER LEARNING

DANE Z 2014



ZASTOSOWANIA

NLP

Konwersja (embedding) słów na wektory:
word2vec, glove, fasttext, BERT

AUDIO/MOWA

Automatic Speech Recognition (ASR) wytrenowane na języku angielskim mogą być z sukcesem wykorzystane do poprawy modeli w innych językach (np. niemieckim)

COMPUTER VISION

CASE STUDY

DOG-CAT CLASSIFICATION

CZYLI JAK ODRÓŻNIĆ PSA OD KOTA?

https://www.kaggle.com/c/dogs-vs-cats/data

DOG-CAT CLASSIFICATION

CZYLI JAK ODRÓŻNIĆ PSA OD KOTA?

https://www.kaggle.com/c/dogs-vs-cats/data

12500

12500

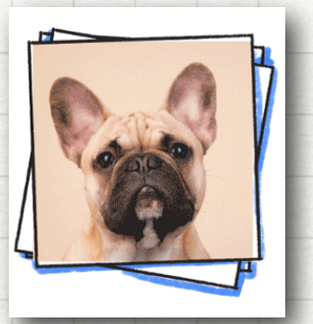
```
files = glob.glob('train/*')

cat_files = [fn for fn in files if 'cat' in fn]
dog_files = [fn for fn in files if 'dog' in fn]
len(cat_files), len(dog_files)
```

tl_1.py hosted with ♥ by GitHub view raw

DOG-CAT CLASSIFICATION

CZYLI JAK ODRÓŻNIĆ PSA OD KOTA?

Wybieramy losowo:
1500 do trenowania
500 do walidacji
500 do testowania



Wybieramy losowo:
1500 do trenowania
500 do walidacji
500 do testowania

```
cat_train = np.random.choice(cat_files, size=1500, replace=False)
dog_train = np.random.choice(dog_files, size=1500, replace=False)
cat_files = list(set(cat_files) - set(cat_train))
dog_files = list(set(dog_files) - set(dog_train))

cat_val = np.random.choice(cat_files, size=500, replace=False)
dog_val = np.random.choice(dog_files, size=500, replace=False)
cat_files = list(set(cat_files) - set(cat_val))
dog_files = list(set(dog_files) - set(dog_val))

cat_test = np.random.choice(cat_files, size=500, replace=False)
dog_test = np.random.choice(dog_files, size=500, replace=False)

print('Cat datasets:', cat_train.shape, cat_val.shape, cat_test.shape)
print('Dog datasets:', dog_train.shape, dog_val.shape, dog_test.shape)
```

tl_2.py hosted with ♥ by GitHub view raw

PRZYGOTOWANIE ZBIORU DANYCH

WCZYTANIE DANYCH

```python
IMG_DIM = (150, 150)

train_files = glob.glob('training_data/*')
train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in train_files]
train_imgs = np.array(train_imgs)
train_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in train_files]

validation_files = glob.glob('validation_data/*')
validation_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in validatic
validation_imgs = np.array(validation_imgs)
validation_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in validation_files]

print('Train dataset shape:', train_imgs.shape,
      '\tValidation dataset shape:', validation_imgs.shape)
```

tl_4.py hosted with ♥ by GitHub view raw

```
Train dataset shape: (3000, 150, 150, 3)
Validation dataset shape: (1000, 150, 150, 3)
```

PRZYGOTOWANIE ZBIORU DANYCH

PRZESKALOWANIE KOLORÓW

```python
IMG_DIM = (150, 150)

train_files = glob.glob('training_data/*')
train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in train_files]
train_imgs = np.array(train_imgs)
train_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in train_files]

validation_files = glob.glob('validation_data/*')
validation_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in validatic
validation_imgs = np.array(validation_imgs)
validation_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in validation_files]

print('Train dataset shape:', train_imgs.shape,
      '\tValidation dataset shape:', validation_imgs.shape)
```

tl_4.py hosted with ♥ by GitHub view raw

```
Train dataset shape: (3000, 150, 150, 3)
Validation dataset shape: (1000, 150, 150, 3)
```

```python
train_imgs_scaled = train_imgs.astype('float32')
validation_imgs_scaled  = validation_imgs.astype('float32')
train_imgs_scaled /= 255
validation_imgs_scaled /= 255

print(train_imgs[0].shape)
array_to_img(train_imgs[0])
```

tl_5.py hosted with ♥ by GitHub view raw

PRZYGOTOWANIE ZBIORU DANYCH

PRZESKALOWANIE KOLORÓW

```python
IMG_DIM = (150, 150)

train_files = glob.glob('training_data/*')
train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in train_files]
train_imgs = np.array(train_imgs)
train_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in train_files]

validation_files = glob.glob('valida
validation_imgs = [img_to_array(load
validation_imgs = np.array(validatio
validation_labels = [fn.split('\\')[

print('Train dataset shape:', train_
      '\tValidation dataset shape:',
```

tl_4.py hosted with ♥ by GitHub

```
Train dataset shape: (3000, 150
Validation dataset shape: (1000
```

```python
train_imgs_scaled = train_imgs.astype('float32')
validation_imgs_scaled  = validation_imgs.astype('float32')
train_imgs_scaled /= 255
validation_imgs_scaled /= 255

print(train_imgs[0].shape)
```

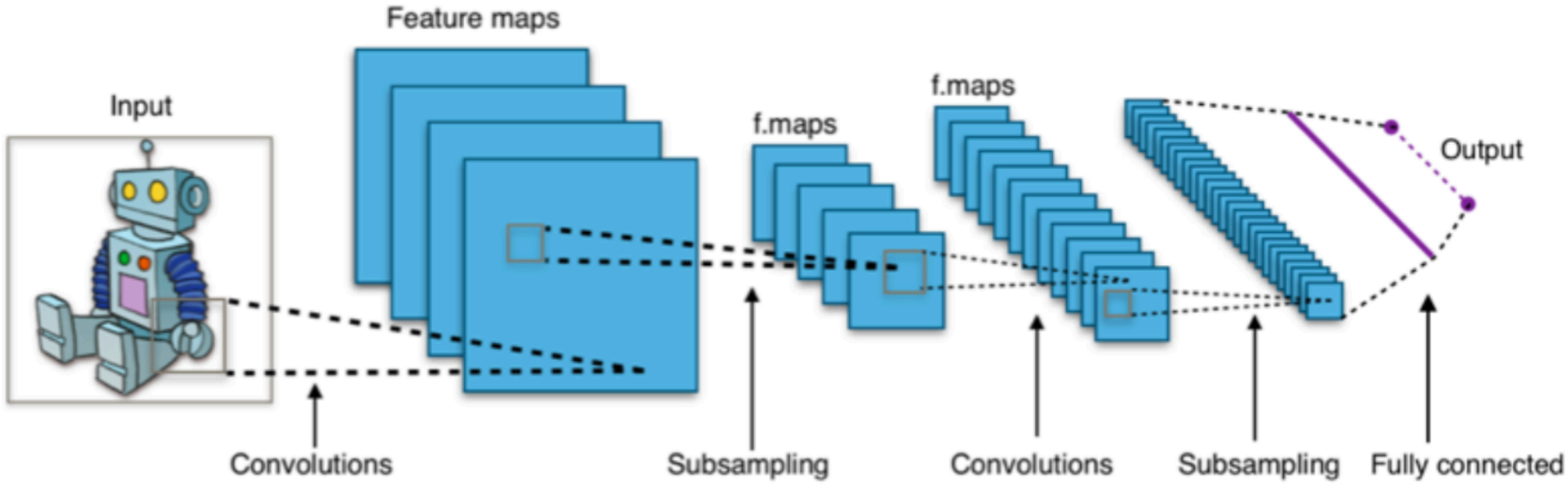
```python
batch_size = 30
num_classes = 2
epochs = 30
input_shape = (150, 150, 3)

# encode text category labels
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(train_labels)
train_labels_enc = le.transform(train_labels)
validation_labels_enc = le.transform(validation_labels)

print(train_labels[1495:1505], train_labels_enc[1495:1505])
```

tl_6.py hosted with ♥ by GitHub view raw

```
['cat', 'cat', 'cat', 'cat', 'cat', 'dog', 'dog', 'dog', 'dog',
'dog'] [0 0 0 0 0 1 1 1 1 1]
```

PROSTY MODEL CNN

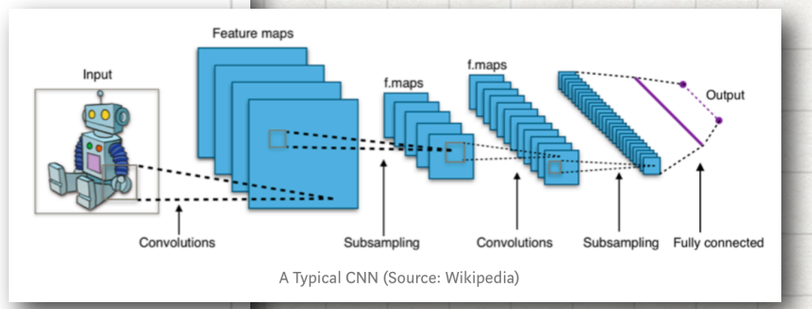
PROSTY MODEL CNN



A Typical CNN (Source: Wikipedia)

PROSTY MODEL CNN

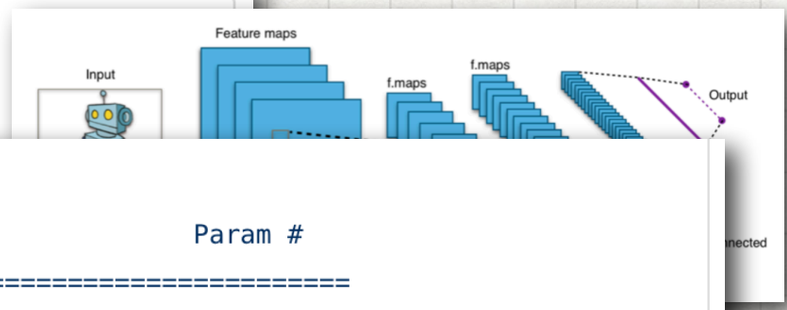
```python
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.models import Sequential
from keras import optimizers

model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(),
              metrics=['accuracy'])

model.summary()
```



A Typical CNN (Source: Wikipedia)

PROSTY MODEL CNN

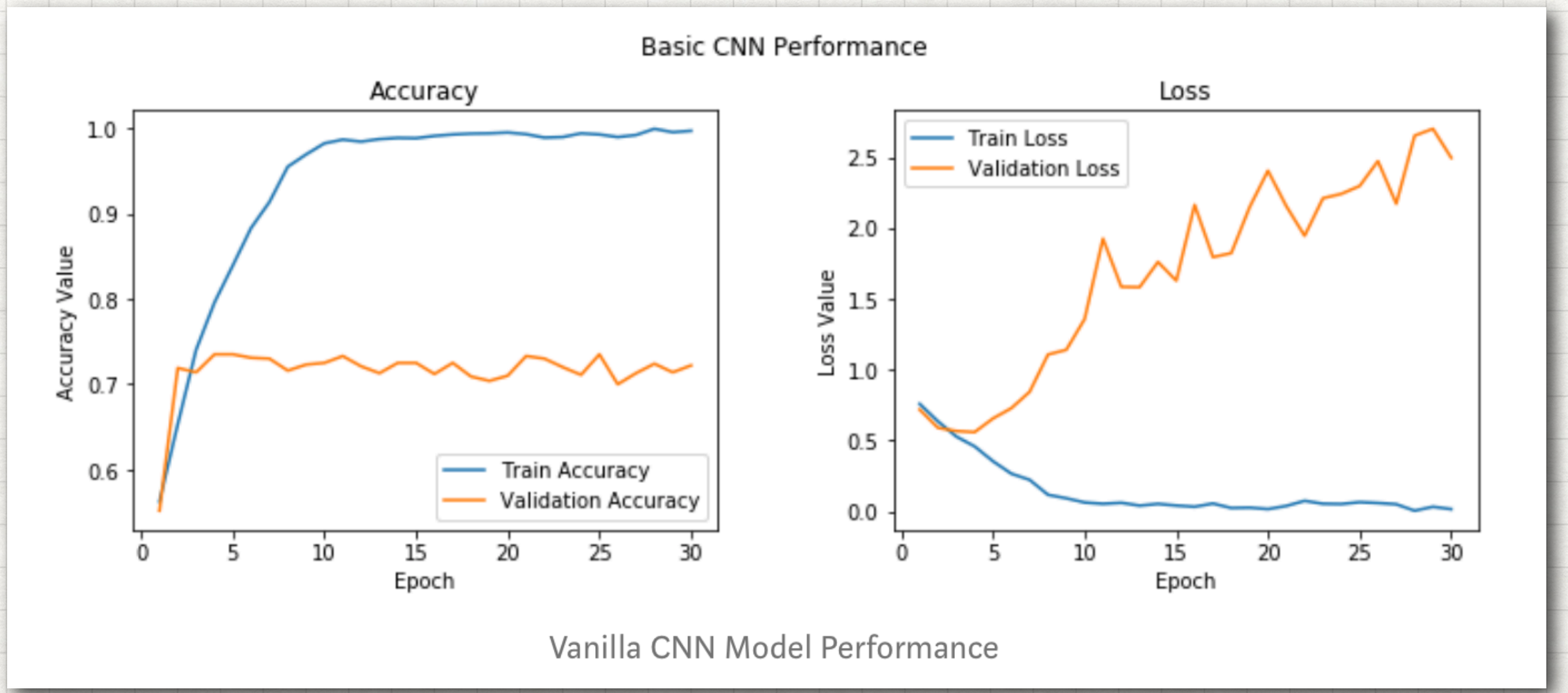
```python
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.models import Sequential
from keras import optimizers

model = Sequential()

model.add(Conv2D(16, kernel_size
                 input_shape=inp
model.add(MaxPooling2D(pool_size

model.add(Conv2D(64, kernel_size
model.add(MaxPooling2D(pool_size

model.add(Conv2D(128, kernel_siz
model.add(MaxPooling2D(pool_size

model.add(Flatten())
model.add(Dense(512, activation=
model.add(Dense(1, activation='s

model.compile(loss='binary_cross
              optimizer=optimize
              metrics=['accuracy

model.summary()
```




```
'''
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        9280
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 36992)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               18940416
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 19,024,513
Trainable params: 19,024,513
Non-trainable params: 0
_________________________________________________________________
'''
```

PROSTY MODEL CNN

WYNIKI

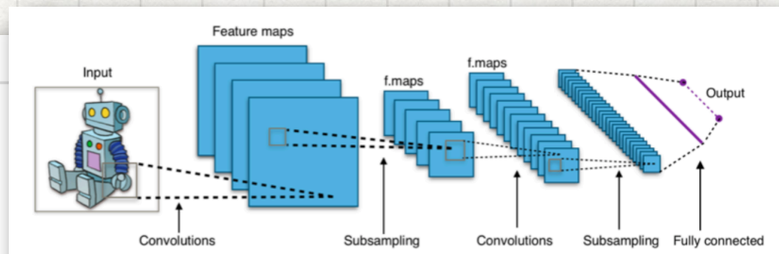


Vanilla CNN Model Performance

CNN Z DROPOUT

PROSTY MODEL CNN

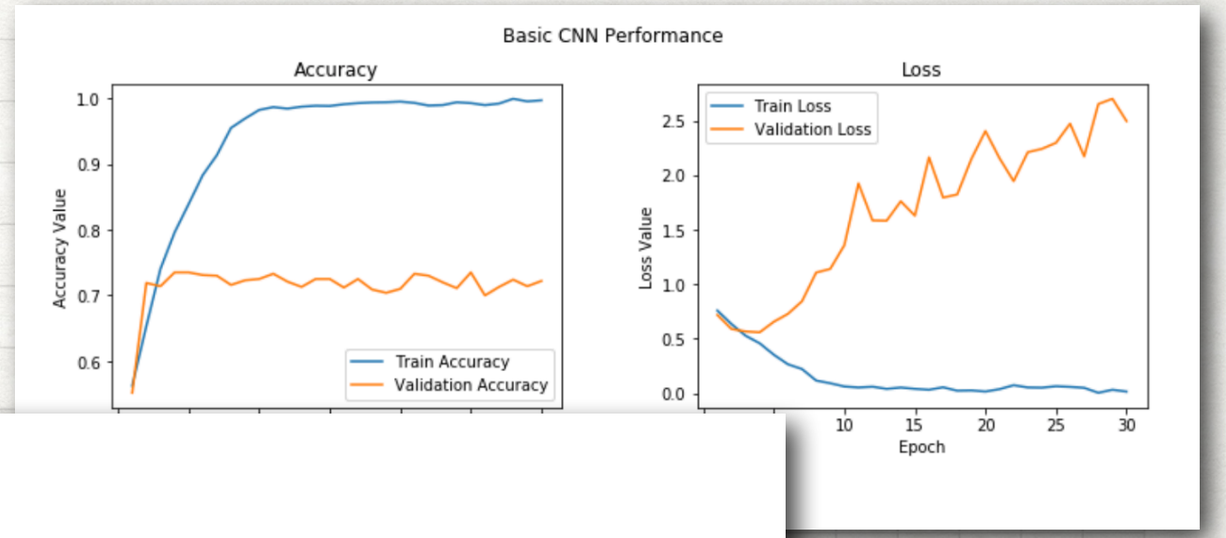
```python
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

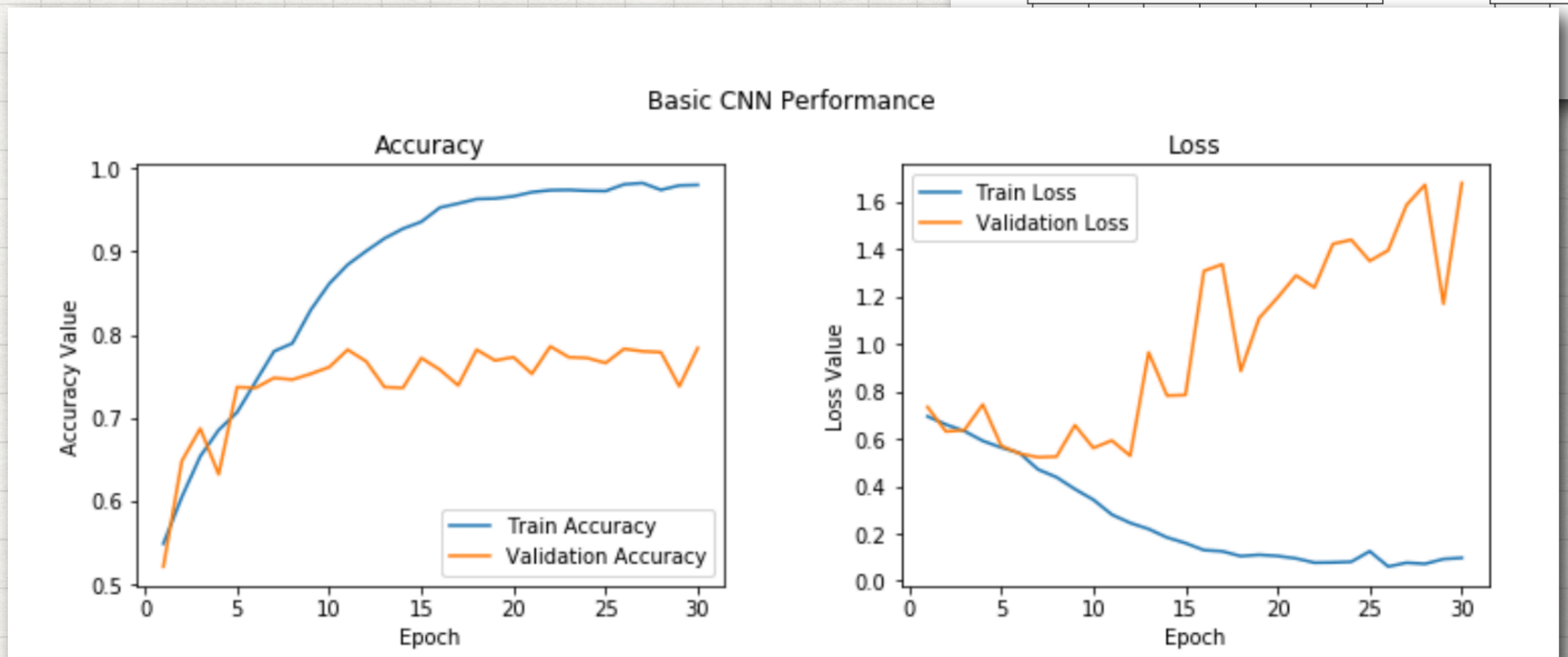


A Typical CNN (Source: Wikipedia)

PROSTY MODEL CNN

WYNIKI



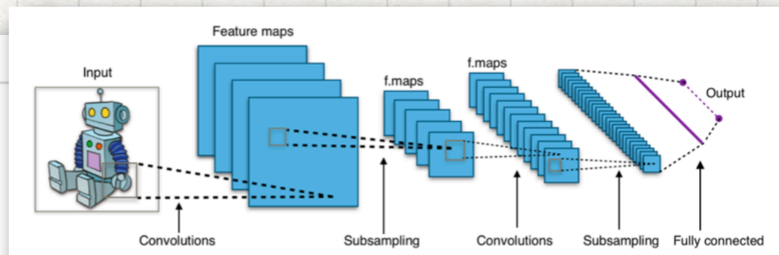


Vanilla CNN Model with Regularization Performance

CNN WITH AUGMENTATION

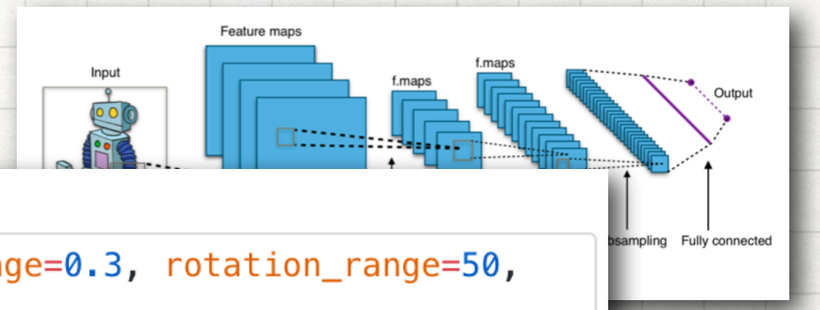
CNN WITH AUGMENTATION

```
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```



A Typical CNN (Source: Wikipedia)

CNN WITH AUGMENTATION



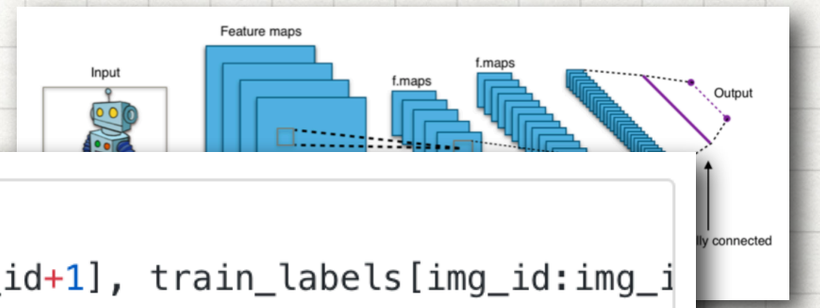

```python
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_r
                                   horizontal_flip=True, fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)
```

tl_11.py hosted with ♥ by GitHub view raw

CNN WITH AUGMENTATION

```python
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_r
                                   horizontal_flip=True, fill_mode='nearest')

val_
```

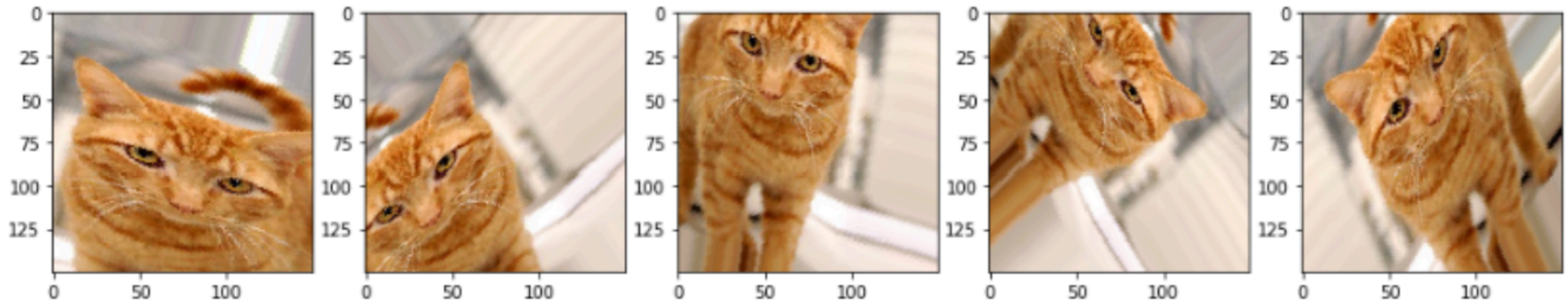
tl_11.py ho




```python
img_id = 2595
cat_generator = train_datagen.flow(train_imgs[img_id:img_id+1], train_labels[img_id:img_i
                                   batch_size=1)
cat = [next(cat_generator) for i in range(0,5)]
fig, ax = plt.subplots(1,5, figsize=(16, 6))
print('Labels:', [item[1][0] for item in cat])
l = [ax[i].imshow(cat[i][0][0]) for i in range(0,5)]
```

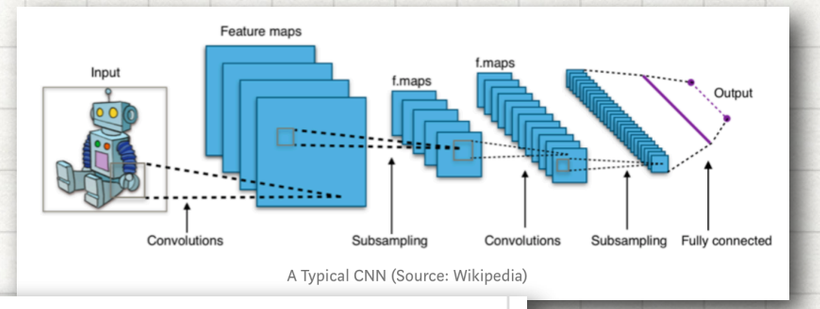
tl_12.py hosted with ❤ by GitHub view raw

```
Labels: ['cat', 'cat', 'cat', 'cat', 'cat']
```



CNN WITH AUGMENTATION

TRENOWANIE



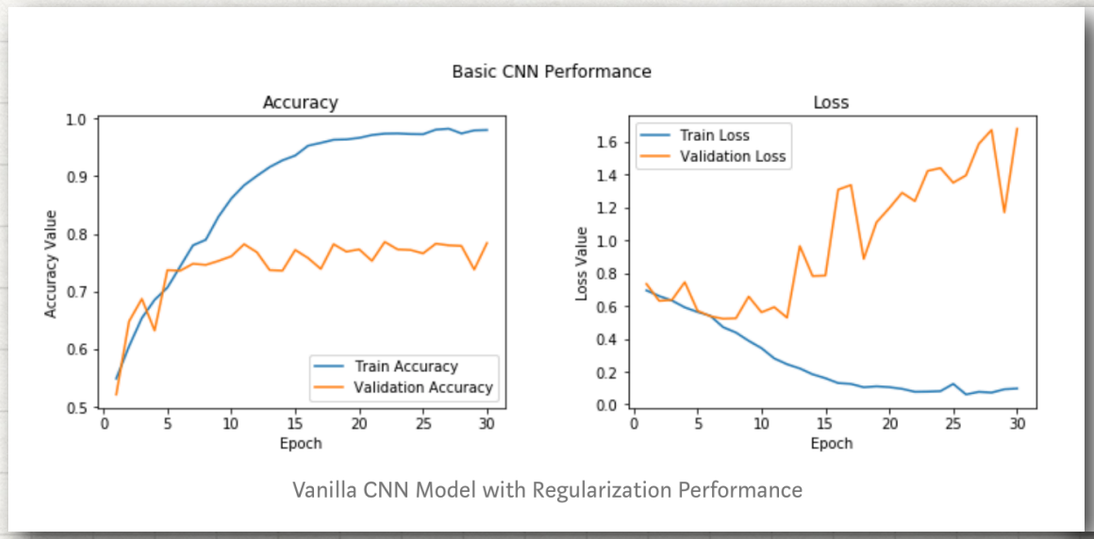
A Typical CNN (Source: Wikipedia)

```
31  model.compile(loss='binary_crossentropy',
32                optimizer=optimizers.RMSprop(lr=1e-4),
33                metrics=['accuracy'])
34
35  history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
36                                validation_data=val_generator, validation_steps=50,
37                                verbose=1)
```

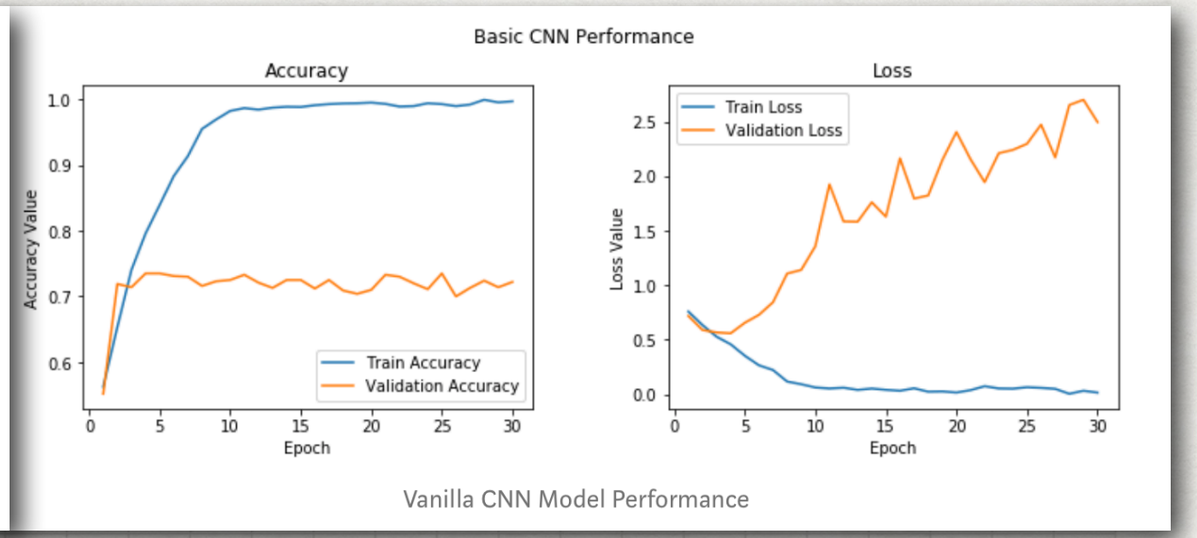
tl_14.py hosted with ♥ by **GitHub** view raw

PROSTY MODEL CNN

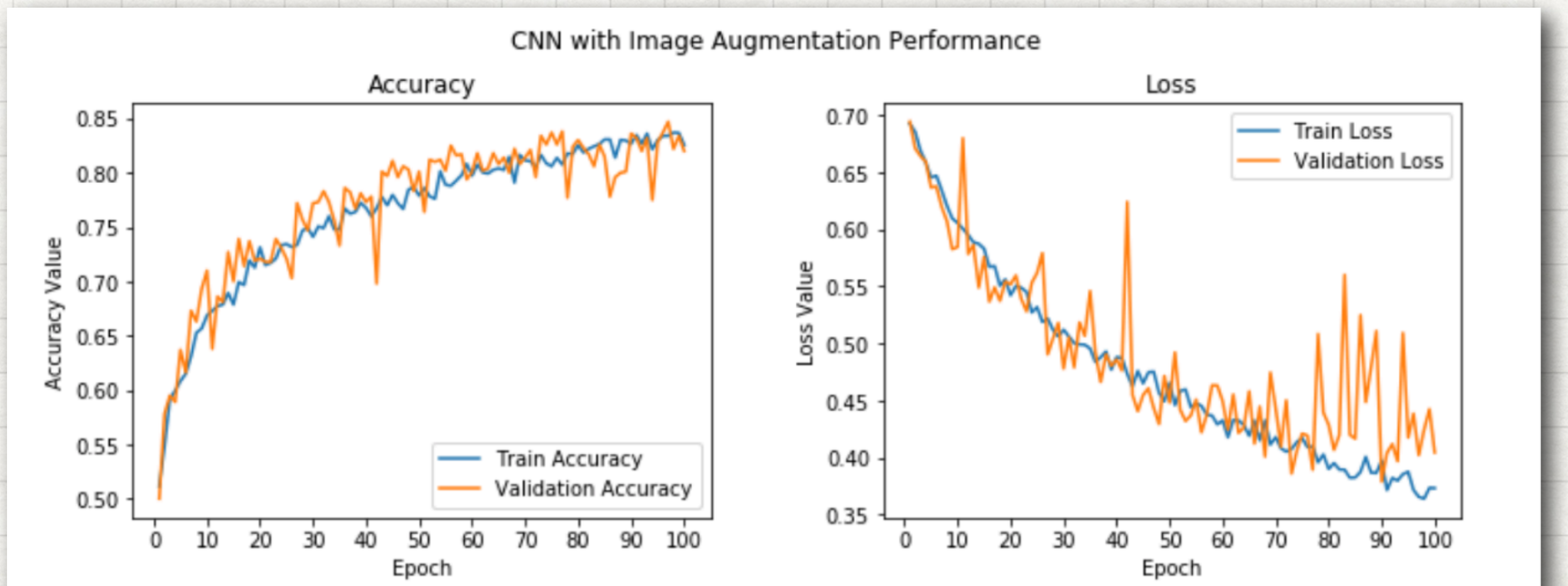
WYNIKI



Vanilla CNN Model with Regularization Performance



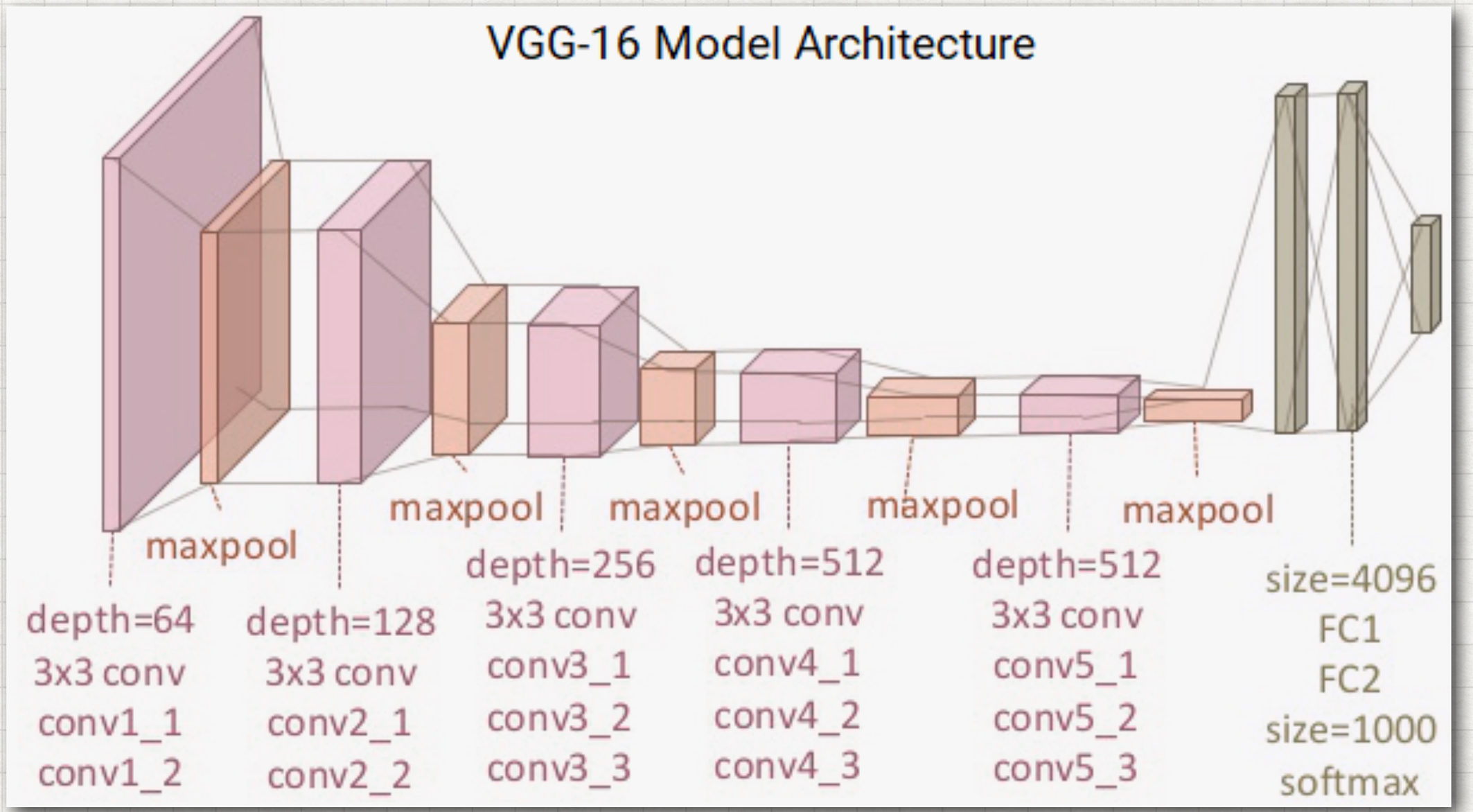
Vanilla CNN Model Performance



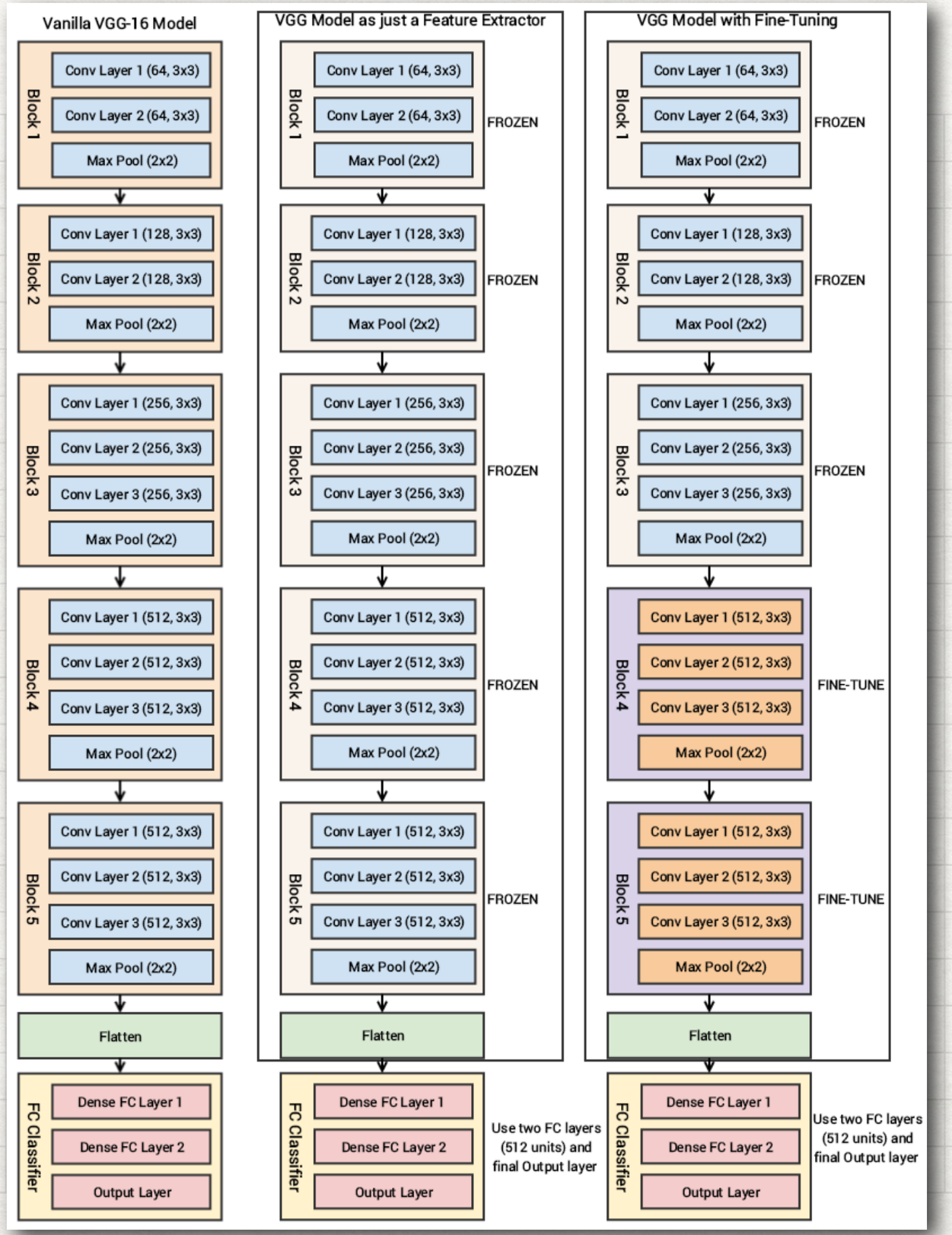
Vanilla CNN Model with Image Augmentation Performance

PRETRAINED MODEL

SIEĆ VGG-16


VGG-16 Model Architecture
maxpool
maxpool
maxpool
maxpool
maxpool
maxpool
depth=64
3x3 conv
conv1_1
conv1_2
depth=128
3x3 conv
conv2_1
conv2_2
depth=256
3x3 conv
conv3_1
conv3_2
conv3_3
depth=512
3x3 conv
conv4_1
conv4_2
conv4_3
depth=512
3x3 conv
conv5_1
conv5_2
conv5_3
size=4096
FC1
FC2
size=1000
softmax

SIEĆ VGG-16


Vanilla VGG-16 Model
VGG Model as just a Feature Extractor
VGG Model with Fine-Tuning
Block 1
Conv Layer 1 (64, 3x3)
Conv Layer 2 (64, 3x3)
Max Pool (2x2)
Block 2
Conv Layer 1 (128, 3x3)
Conv Layer 2 (128, 3x3)
Max Pool (2x2)
Block 3
Conv Layer 1 (256, 3x3)
Conv Layer 2 (256, 3x3)
Conv Layer 3 (256, 3x3)
Max Pool (2x2)
Block 4
Conv Layer 1 (512, 3x3)
Conv Layer 2 (512, 3x3)
Conv Layer 3 (512, 3x3)
Max Pool (2x2)
Block 5
Conv Layer 1 (512, 3x3)
Conv Layer 2 (512, 3x3)
Conv Layer 3 (512, 3x3)
Max Pool (2x2)
Flatten
FC Classifier
Dense FC Layer 1
Dense FC Layer 2
Output Layer
FROZEN
FINE-TUNE
Use two FC layers (512 units) and final Output layer

VGG-16

GOTOWY MODEL

```

from keras.applications import vgg16
from keras.models import Model
import keras

vgg = vgg16.VGG16(include_top=False, weights='imagenet',
                                     input_shape=input_shape)

output = vgg.layers[-1].output
output = keras.layers.Flatten()(output)
vgg_model = Model(vgg.input, output)

vgg_model.trainable = False
for layer in vgg_model.layers:
    layer.trainable = False

import pandas as pd
pd.set_option('max_colwidth', -1)
layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])
```

tl_15.py hosted with ♥ by GitHub view raw

VGG-16

GOTOWE *FEATURES*

```python
def get_bottleneck_features(model, input_imgs):
    features = model.predict(input_imgs, verbose=0)
    return features

train_features_vgg = get_bottleneck_features(vgg_model, train_imgs_scaled)
validation_features_vgg = get_bottleneck_features(vgg_model, validation_imgs_scaled)

print('Train Bottleneck Features:', train_features_vgg.shape,
      '\tValidation Bottleneck Features:', validation_features_vgg.shape)
```

tl_16.py hosted with ♥ by GitHub view raw

```
Train Bottleneck Features: (3000, 8192)
Validation Bottleneck Features: (1000, 8192)
```

VGG-16 TRANSFER

CZĘŚĆ KLASYFIKUJĄCA

```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
from keras.models import Sequential
from keras import optimizers

input_shape = vgg_model.output_shape[1]

model = Sequential()
model.add(InputLayer(input_shape=(input_shape,)))
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

VGG-16 TRANSFER

CZĘŚĆ KLASYFIKUJĄCA

```python
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
from keras.models import Sequential
from keras import optimizers

input_shape = vgg_model.output_shape[1]
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,457,985
Trainable params: 4,457,985
Non-trainable params: 0
_________________________________________________________________
```

VGG-16 TRANSFER

TRENOWANIE

```python
def get_bottleneck_features(model, input_imgs):
    features = model.predict(input_imgs, verbose=0)
    return features

train_features_vgg = get_bottleneck_features(vgg_model, train_imgs_scaled)
validation_features_vgg = get_bottleneck_features(vgg_model, validation_imgs_scaled)

print('Train Bottleneck Features:', train_features_vgg.shape,
      '\tValidation Bottleneck Features:', validation_features_vgg.shape)
```

tl_16.py hosted with ♥ by GitHub view raw

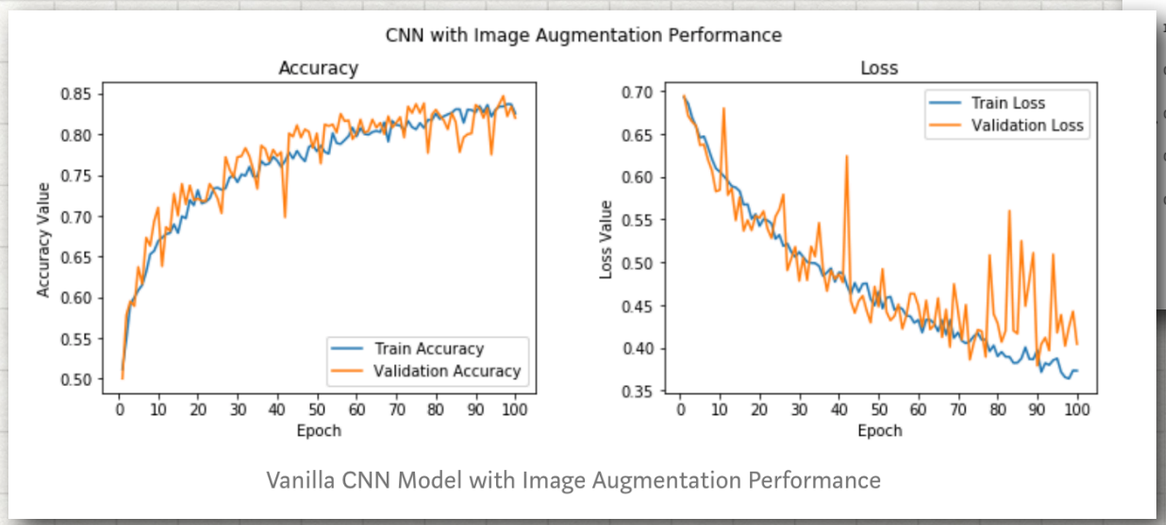
```
Train Bottleneck Features: (3000, 8192)
Validation Bott
```

```python
history = model.fit(x=train_features_vgg, y=train_labels_enc,
                    validation_data=(validation_features_vgg, validation_labels_enc),
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)
```

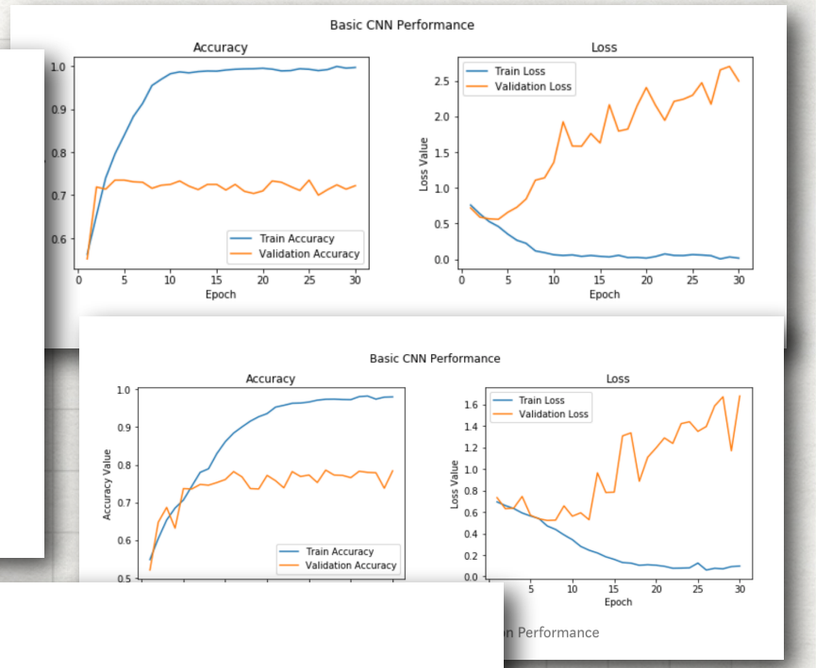
tl_18.py hosted with ♥ by GitHub view raw

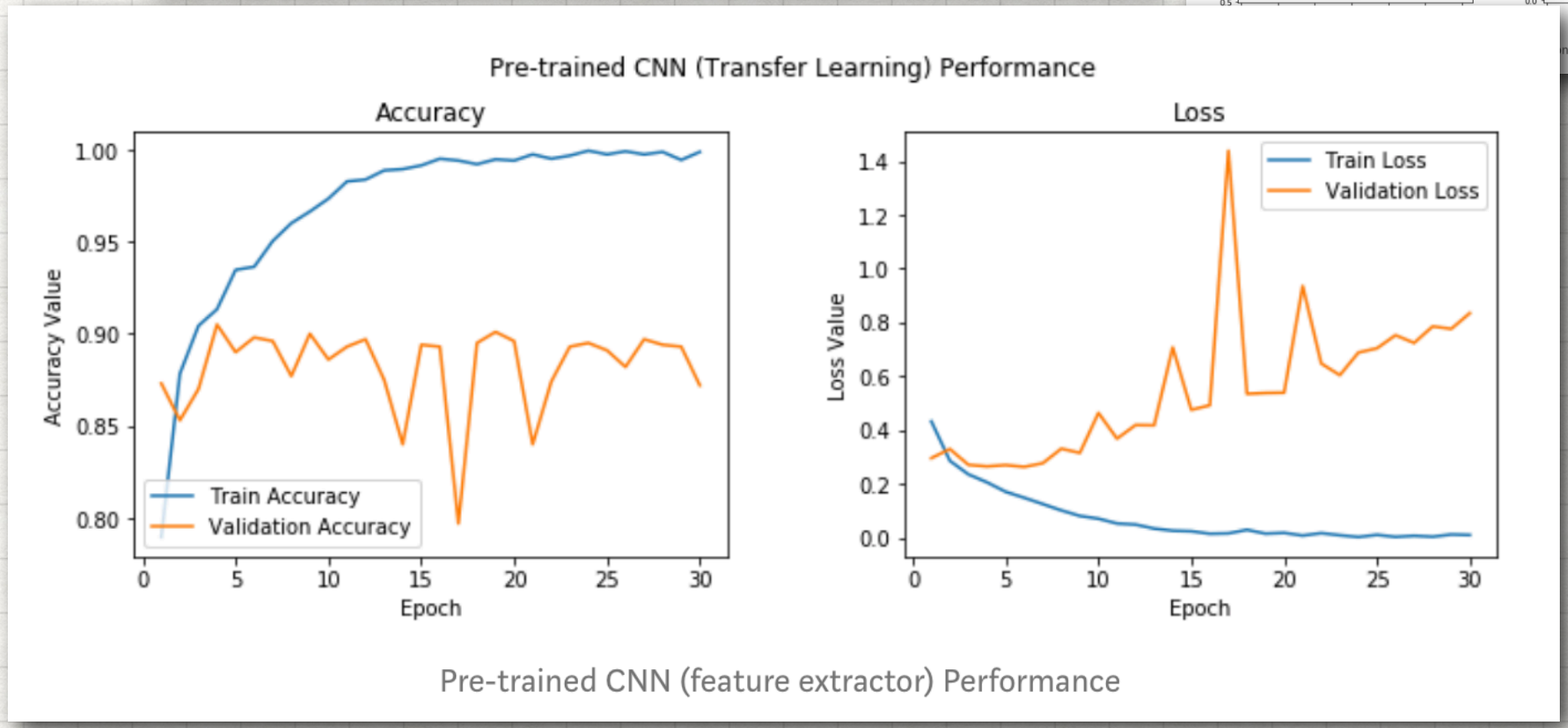
VGG-16 TRANSFER

WYNIKI



Vanilla CNN Model with Image Augmentation Performance





Pre-trained CNN (feature extractor) Performance

VGG-16 FEATURE EXTRACTOR WITH AUGMENTATION

GENERATOR OBRAZÓW

PRZYPOMNIENIE

```
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_r
                                   horizontal_flip=True, fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)
```

tl_19.py hosted with ♥ by GitHub view raw

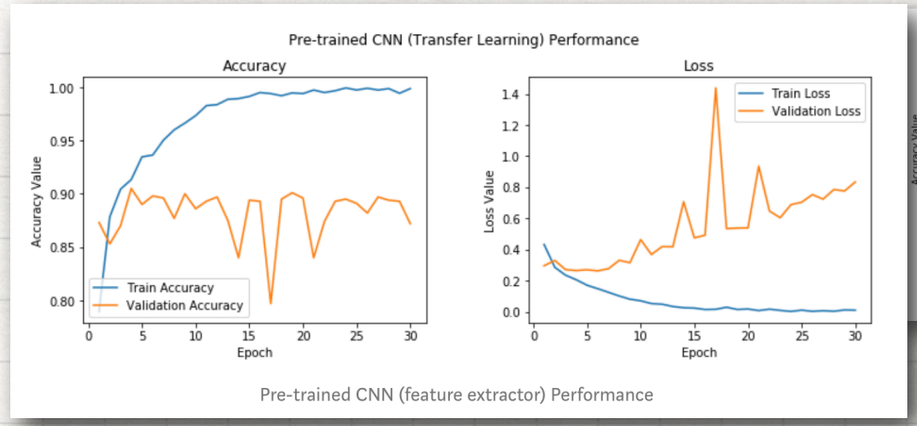
VGG-16 FEATURE EXTRACTION

WITH AUGMENTATION

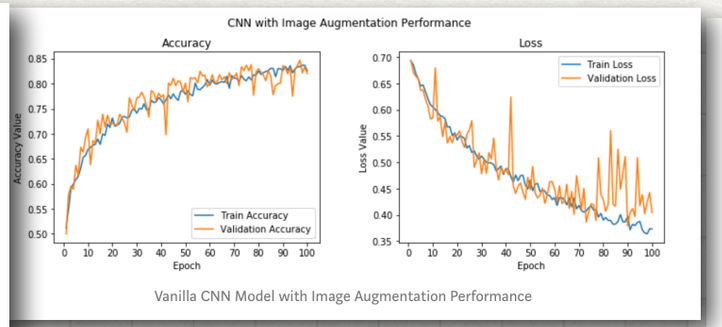
```python
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
from keras.models import Sequential
from keras import optimizers

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['accuracy'])

history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
                              validation_data=val_generator, validation_steps=50,
                              verbose=1)
```

VGG-16 TRANSFER

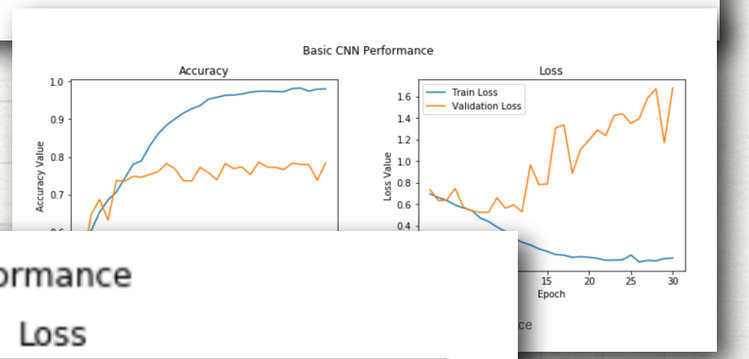
WYNIKI

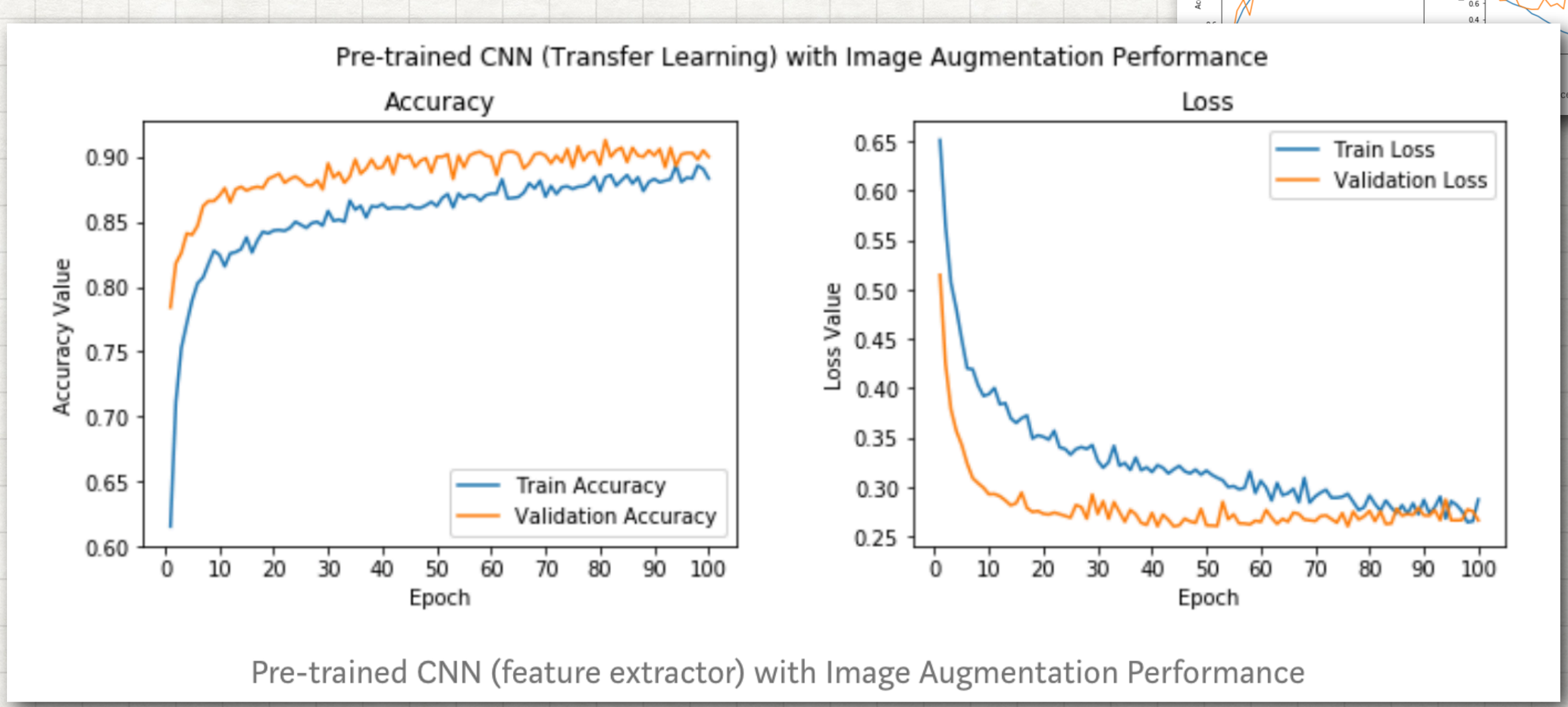


Pre-trained CNN (feature extractor) Performance



Vanilla CNN Model with Image Augmentation Performance





Pre-trained CNN (feature extractor) with Image Augmentation Performance

VGG-16 FINE TUNING WITH AUGMENTATION

VGG-16 FINE TUNING

WITH AUGMENTATION

```
vgg_model.trainable = True

set_trainable = False
for layer in vgg_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])
```

tl_21.py hosted with ♥ by GitHub view raw

VGG-16 FEATURE EXTRACTION

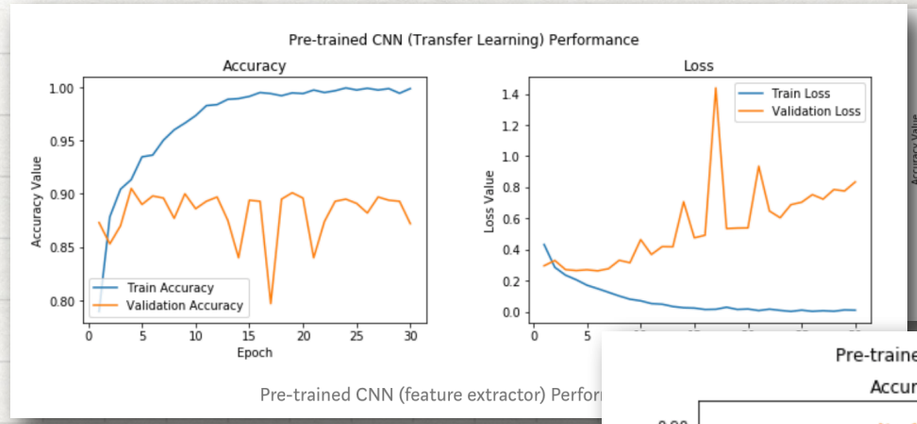
WITH AUGMENTATION

```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
from keras.models import Sequential
from keras import optimizers

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])

history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
                              validation_data=val_generator, validation_steps=50,
                              verbose=1)
```

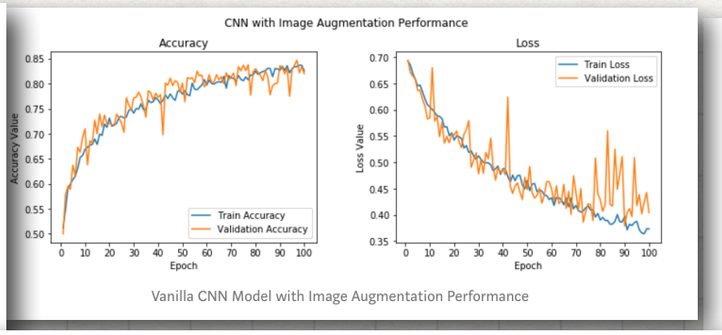
tl_22.py hosted with ♥ by GitHub view raw

VGG-16 FINE-TUNING

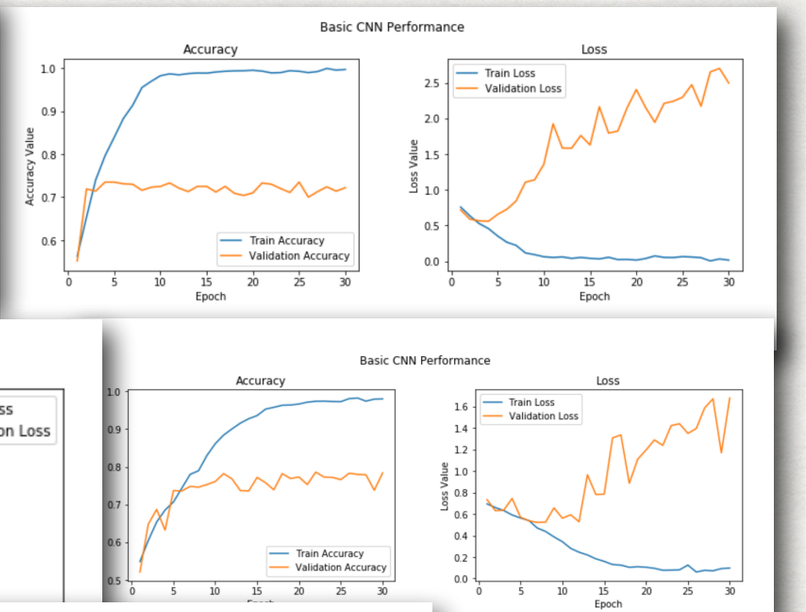
WITH AUGMENTATION

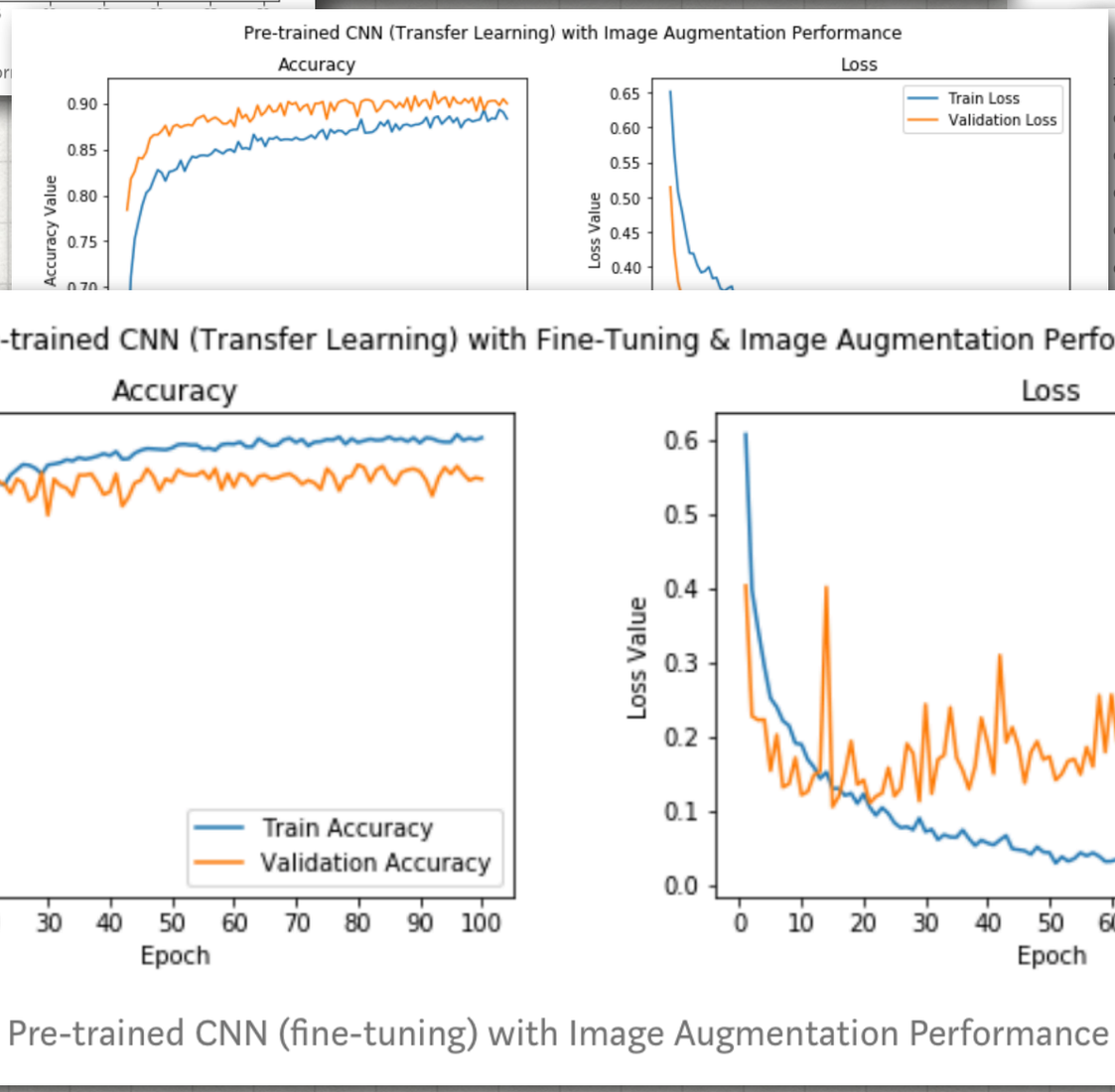


Pre-trained CNN (feature extractor) Perfor



Vanilla CNN Model with Image Augmentation Performance





ularization Performance

Pre-trained CNN (fine-tuning) with Image Augmentation Performance

ACCURACY

PODSUMOWANIE

- Simple CNN: 72%
- CNN with dropout: 78%
- CNN with augmentation: 82%
- VGG-16 pretrained: 88%
- VGG-16 feature extraction with augmentation: 90%
- VGG-16 fine-tuning with augmentation: 95%

BIBLIOGRAFIA

- wykład w dużej mierze oparty o: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

- https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

- Sinno Jialin Pan and Qiang Yang, *A Survey on Transfer Learning*