Implementacja sieci neuronowej

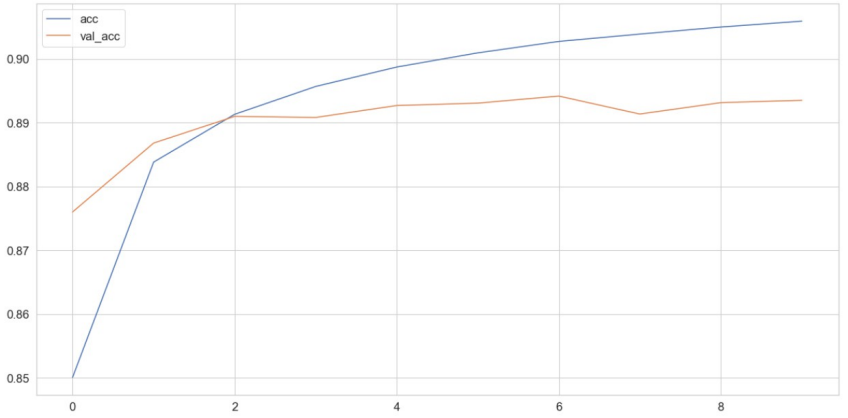
In [22]:
```
def getModel():
    model = tf.keras.models.Sequential([
                tf.keras.layers.Conv2D(8,(3,3), activation='relu',input_shape=(28, 28,1)),
                tf.keras.layers.Conv2D(16,(3,3), activation='relu'),
                tf.keras.layers.Conv2D(32,(3,3), activation='relu'),
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dense(10, activation='softmax')])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
model = getModel()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 8)         80
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 16)        1168
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 32)        4640
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 10)                154890
=================================================================
Total params: 160,778
Trainable params: 160,778
Non-trainable params: 0
_________________________________________________________________
```

In [24]:
```
epochs=10
batch_size=256

history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
                    validation_data=(X_val,y_val),)
```

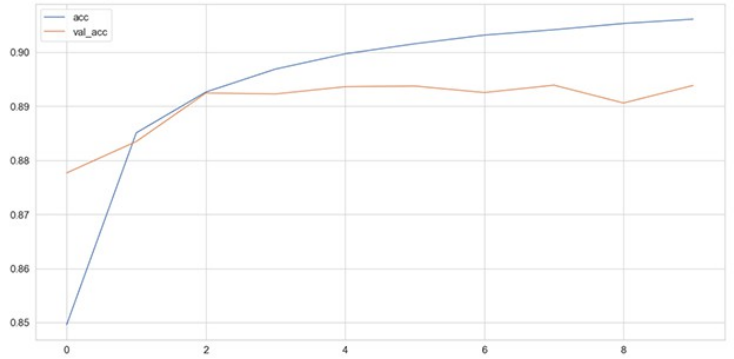
In [25]:
```
plt.figure(figsize=(20,10))
plt.plot(history.history['accuracy'], label='acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x12beac56310>

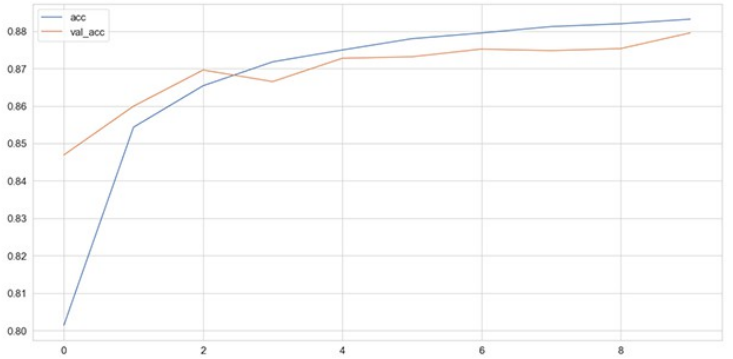


Optymalizacja – zmiana funkcji aktywacji

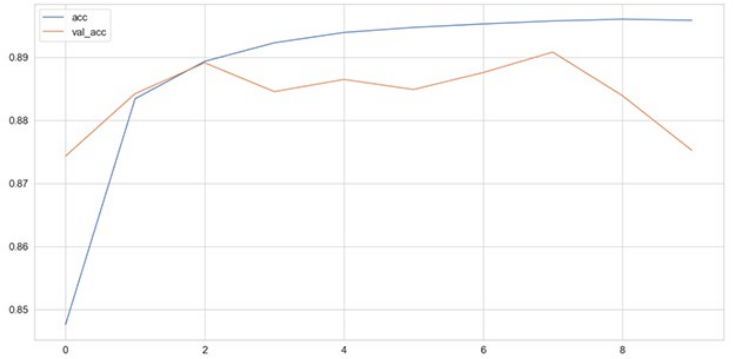
Relu + Adam



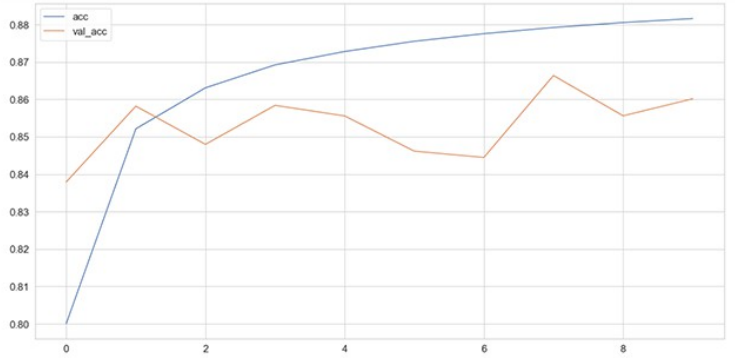
Swish + Adam



Relu + RMSprop



Swish + RMSprop



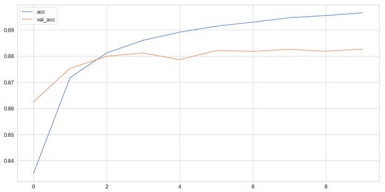
Optymalizacja - Dropout

In [26]:
```
def getModelDropout(ratios=[0.0,0.0]):
    model = tf.keras.models.Sequential([
                tf.keras.layers.Conv2D(8,(3,3), activation='relu',input_shape=(28, 28,1)),
                tf.keras.layers.Dropout(ratios[0]),
                tf.keras.layers.Conv2D(32,(3,3), activation='relu'),
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dropout(ratios[1]),
                tf.keras.layers.Dense(10, activation='softmax')])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
model = getModelDropout([0,0])
model.summary()
#del model
```

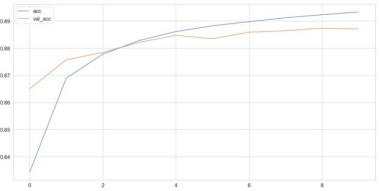
```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 8)         80
_________________________________________________________________
dropout_4 (Dropout)          (None, 26, 26, 8)         0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 24, 24, 32)        2336
_________________________________________________________________
flatten_8 (Flatten)          (None, 18432)             0
_________________________________________________________________
dropout_5 (Dropout)          (None, 18432)             0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                184330
=================================================================
Total params: 186,746
Trainable params: 186,746
Non-trainable params: 0
_________________________________________________________________
```

Dropout

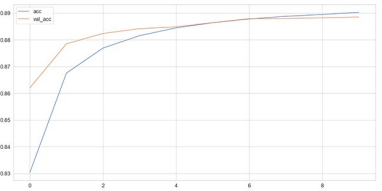
ratio = 0.0, 0.0



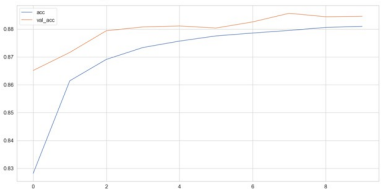
ratio = 0.0, 0.1



ratio = 0.0, 0.2



ratio = 0.1, 0.2



Optymalizacja – Keras Tuner

In [6]:
```python
from keras_tuner import RandomSearch

def getModel(hp):

    l1_l2=tf.keras.regularizers.l1_l2(hp.Float('L1_L2', 1e-8, 0.01, sampling='log'))
    l =[tf.keras.layers.Conv2D(8,(3,3), padding='same', activation='relu',
                               kernel_regularizer=l1_l2, input_shape=(img_size, img_size, 3)),]
    pooling = hp.Int('avg_pooling',0,3)
    bn=hp.Int('BN',0,2)

    if bn>0:
        l.append(tf.keras.layers.BatchNormalization())
    for i in range(hp.Int('convs',0,4)):
        if pooling==1:
            l.append(tf.keras.layers.AvgPool2D())
        if pooling>=2:
            l.append(tf.keras.layers.MaxPool2D())
        l.append(tf.keras.layers.Conv2D(2**(5+i),(3,3), padding='same', activation='relu',
                         kernel_regularizer=l1_l2))
    if bn==2:
        l.append(tf.keras.layers.BatchNormalization())
    l.append(tf.keras.layers.GlobalAveragePooling2D())
    l.append(tf.keras.layers.Dropout(hp.Float('dropout',0,0.9)))
    l.append(tf.keras.layers.Dense(num_classes, activation='softmax'))

    model = tf.keras.models.Sequential(l)
    model.compile(optimizer=tf.keras.optimizers.Adam(hp.Float('lr',0.000001, 0.01, sampling='log')),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

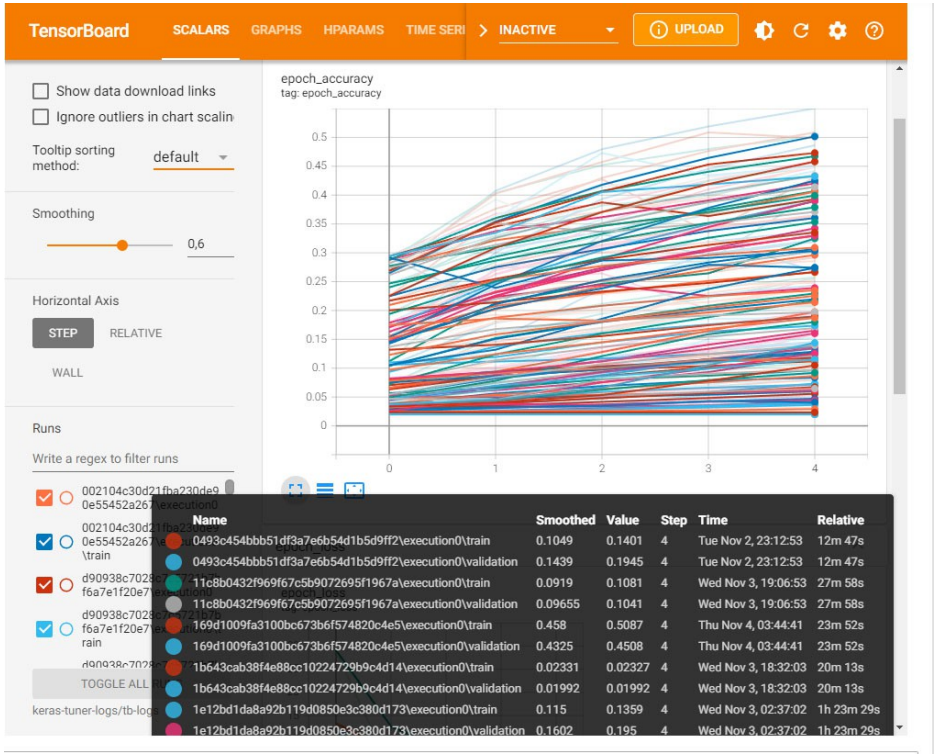
In [7]:
```python
tuner =RandomSearch(getModel,
                    objective="val_accuracy",
                    max_trials=15,
                    executions_per_trial=3,  # Uśrednianie wyników
                    overwrite=True,
                    directory="ktuner",
                    project_name="resisc45")
```

In [9]:
```python
tuner.search_space_summary()
```

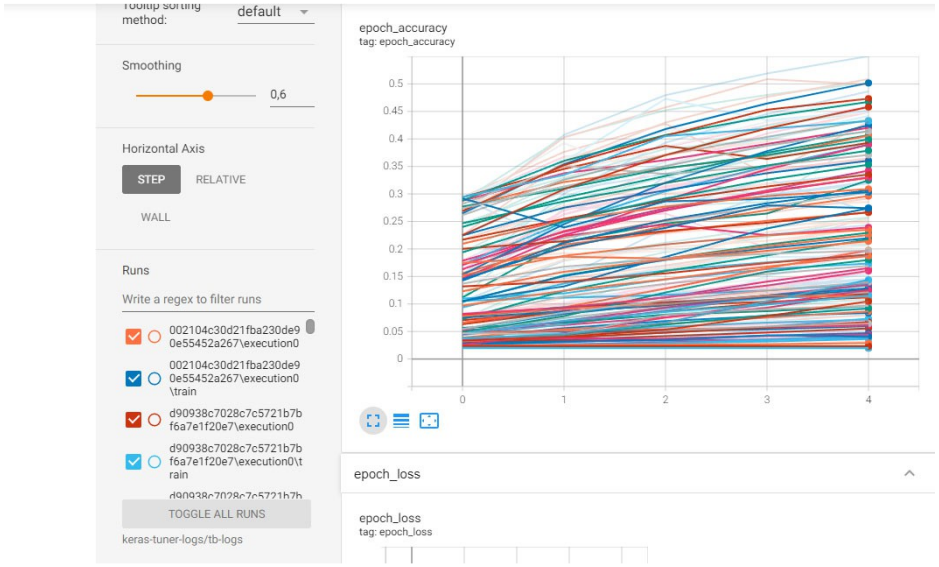
```
Search space summary
Default search space size: 6
L2 (Float)
{'default': 1e-08, 'conditions': [], 'min_value': 1e-08, 'max_value': 0.01, 'step': None, 'sampling': 'log'}
avg_pooling (Int)
{'default': None, 'conditions': [], 'min_value': 0, 'max_value': 2, 'step': 1, 'sampling': None}
BN (Int)
{'default': None, 'conditions': [], 'min_value': 0, 'max_value': 2, 'step': 1, 'sampling': None}
convs (Int)
{'default': None, 'conditions': [], 'min_value': 0, 'max_value': 3, 'step': 1, 'sampling': None}
dropout (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.9, 'step': None, 'sampling': None}
lr (Float)
{'default': 1e-05, 'conditions': [], 'min_value': 1e-05, 'max_value': 0.1, 'step': None, 'sampling': 'log'}
```

In [1]:
```python
from torch.utils import tensorboard
```

In [2]:
```python
%reload_ext tensorboard
%tensorboard --logdir keras-tuner-logs/tb-logs --port 6001
```



Optymalizacja – Keras Tuner

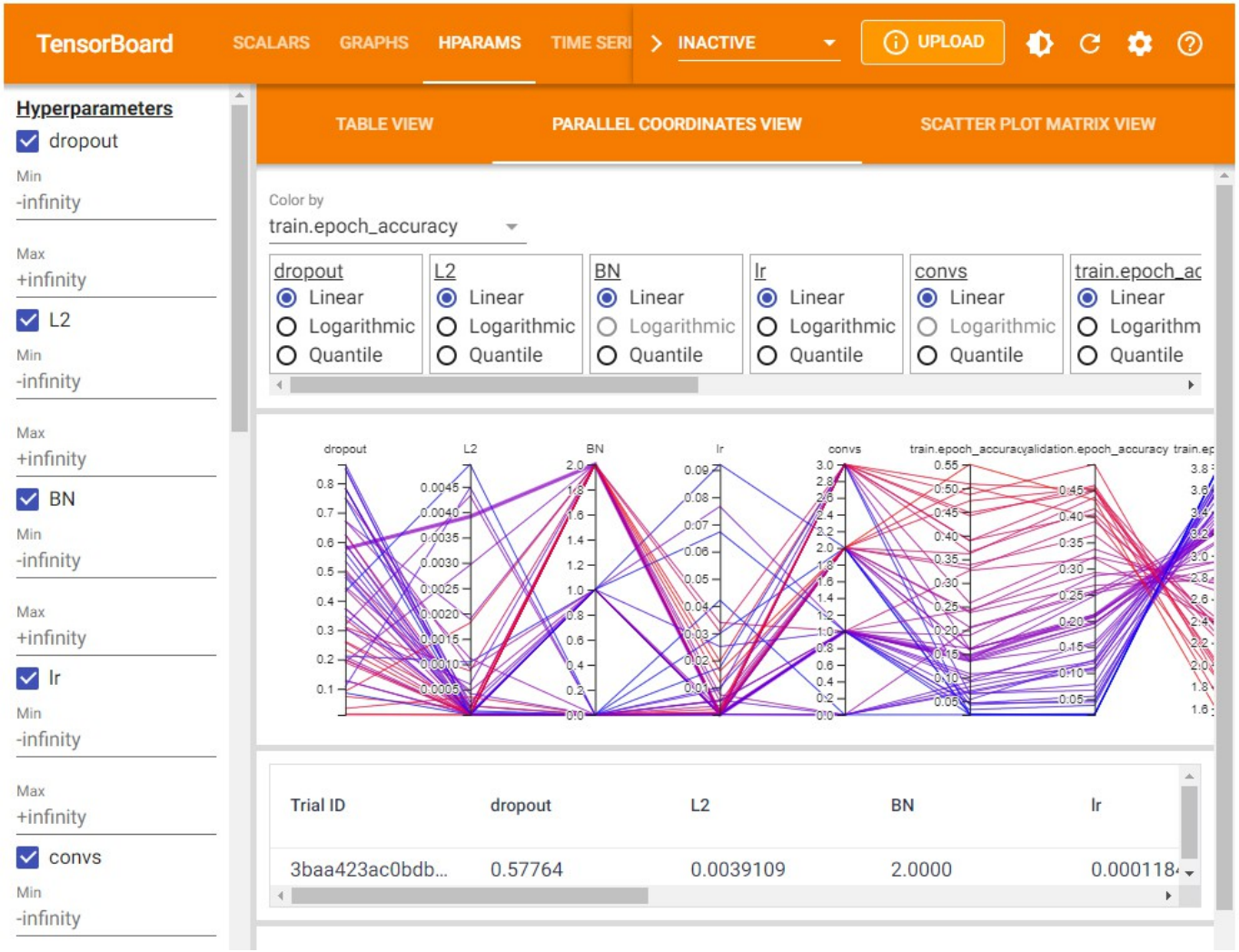



```
In [11]: tuner.search(ds_train, epochs=5, validation_data=(ds_val),
                     callbacks=[tf.keras.callbacks.TensorBoard(log_dir='keras-tuner-logs/tb-logs', write_graph=True)])

Trial 50 Complete [00h 15m 46s]
val_accuracy: 0.19735169410705566

Best val_accuracy So Far: 0.5086864233016968
Total elapsed time: 15h 13m 45s
INFO:tensorflow:Oracle triggered exit
```

```
In [13]: models[0].summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 8)       224
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 8)       32
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      2336
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 32)      128
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 45)                1485
=================================================================
Total params: 4,205
Trainable params: 4,125
Non-trainable params: 80
_________________________________________________________________
```

```
In [14]: models[1].summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 8)       224
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 8)       32
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 32)      2336
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 256, 256, 128)     73856
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 45)                5805
=================================================================
Total params: 100,749
Trainable params: 100,733
Non-trainable params: 16
_________________________________________________________________
```

Optymalizacja – Keras Tuner



Optymalizacja - Optuna

In [6]:
```
import optuna
from optuna.trial import TrialState
from optuna.integration import TFKerasPruningCallback, TensorBoardCallback


def getModel(lr, pooling, flatten, convs):
    l = [tf.keras.layers.Conv2D(8,(3,3), padding='same', activation='relu',input_shape=(img_size, img_size,3)),]
    for i in range(int(convs)):
        if pooling=='avg':
            l.append(tf.keras.layers.AvgPool2D())
        if pooling=='max':
            l.append(tf.keras.layers.MaxPool2D())
        l.append(tf.keras.layers.Conv2D(2**(i+4),(3,3), padding='same', activation='relu'))
    if flatten=='global':
        l.append(tf.keras.layers.GlobalAveragePooling2D())
    else:
        l.append(tf.keras.layers.Flatten())
    l.append(tf.keras.layers.Dense(num_classes, activation='softmax'))

    model = tf.keras.models.Sequential(l)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

params={'lr':0.001,
        'pooling':'avg',
        'flatten':'global',
        'convs':5}

model = getModel(**params)
model.summary()
del model
```

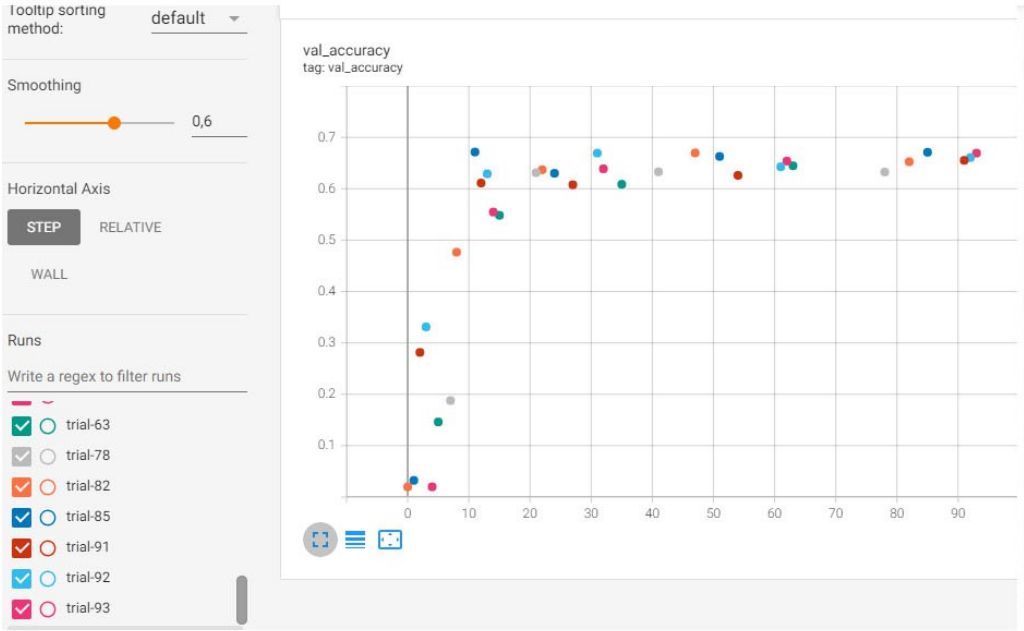
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 8)       224
_________________________________________________________________
average_pooling2d (AveragePo (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
average_pooling2d_1 (Average (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
average_pooling2d_2 (Average (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
average_pooling2d_3 (Average (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
average_pooling2d_4 (Average (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
global_average_pooling2d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 45)                11565
=================================================================
Total params: 405,117
Trainable params: 405,117
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
epochs = 20
```

In [8]:
```
def objective(trial):
    convs = trial.suggest_int("convs", 0, 3)
    pooling = trial.suggest_categorical("pooling", ["no","avg", "max"])
    flatten = trial.suggest_categorical("flatten", ["global", "flatten"])
    lr = trial.suggest_float("lr", 1e-5, 1e-1, log=True)

    model = getModel(lr, pooling, flatten, convs)

    history = model.fit(ds_train, # 10% of data
                        epochs=epochs,
                        batch_size=batch_size,
                        verbose=0,
                        validation_freq=1,
                        validation_data=(ds_val),
                        callbacks=[TFKerasPruningCallback(trial, "val_accuracy")])

    return history.history['val_accuracy'][-1] # return last val acc
```

In [9]:
```
study = optuna.create_study(direction="maximize")  # ==> maximize accuracy
study.optimize(objective, n_trials=100, timeout=None,
               callbacks=[TensorBoardCallback('optuna-keras-logs2','val_accuracy')])

pruned_trials = study.get_trials(deepcopy=False, states=[TrialState.PRUNED])
complete_trials = study.get_trials(deepcopy=False, states=[TrialState.COMPLETE])

print("Study statistics: ")
print("  Number of finished trials: ", len(study.trials))
print("  Number of pruned trials: ", len(pruned_trials))
print("  Number of complete trials: ", len(complete_trials))

print("Best trial:")
trial = study.best_trial

print("  Value: ", trial.value)

print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

Optymalizacja - Optuna




```
atten': 'global', 'lr': 0.0016557650763173051}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 08:48:41,387] Trial 62 finished with value: 0.6537076234817505 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0014974906870220015}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 09:57:26,664] Trial 63 finished with value: 0.6447033882141113 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0017040832215024272}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 10:04:20,063] Trial 64 pruned. Trial was pruned at epoch 1.
[I 2021-12-02 10:07:46,105] Trial 65 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 10:41:51,409] Trial 66 pruned. Trial was pruned at epoch 9.
[I 2021-12-02 11:20:00,827] Trial 67 pruned. Trial was pruned at epoch 10.
[I 2021-12-02 11:22:47,409] Trial 68 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 11:53:45,901] Trial 69 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 11:57:14,581] Trial 70 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 12:00:43,116] Trial 71 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 12:27:54,834] Trial 72 pruned. Trial was pruned at epoch 7.
[I 2021-12-02 12:31:17,913] Trial 73 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 12:34:37,388] Trial 74 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 12:50:52,367] Trial 75 pruned. Trial was pruned at epoch 4.
[I 2021-12-02 12:57:25,822] Trial 76 pruned. Trial was pruned at epoch 1.
[I 2021-12-02 13:00:47,367] Trial 77 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 14:05:32,490] Trial 78 finished with value: 0.6325212121009827 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0011569458891214 95}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 14:08:48,621] Trial 79 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 14:11:47,315] Trial 80 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 14:15:04,002] Trial 81 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 15:20:33,607] Trial 82 finished with value: 0.6525423526763916 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0015587859065636276}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 15:23:51,562] Trial 83 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 15:30:19,544] Trial 84 pruned. Trial was pruned at epoch 1.
[I 2021-12-02 16:34:43,643] Trial 85 finished with value: 0.6709745526313782 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0027921673799563237}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 16:38:01,528] Trial 86 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 17:00:56,823] Trial 87 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 17:11:03,914] Trial 88 pruned. Trial was pruned at epoch 2.
[I 2021-12-02 17:14:27,161] Trial 89 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 17:17:49,497] Trial 90 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 18:24:58,778] Trial 91 finished with value: 0.6551907062530518 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0020396384183621755}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 19:32:01,470] Trial 92 finished with value: 0.6605932116508484 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0019447596710891887}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 20:40:31,640] Trial 93 finished with value: 0.6691737174987793 and parameters: {'convs': 3, 'pooling': 'avg', 'fl
atten': 'global', 'lr': 0.0020273400701534407}. Best is trial 11 with value: 0.6713982820510864.
[I 2021-12-02 21:01:13,136] Trial 94 pruned. Trial was pruned at epoch 5.
[I 2021-12-02 21:14:33,838] Trial 95 pruned. Trial was pruned at epoch 3.
[I 2021-12-02 21:44:30,326] Trial 96 pruned. Trial was pruned at epoch 8.
[I 2021-12-02 21:47:43,820] Trial 97 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 21:50:16,466] Trial 98 pruned. Trial was pruned at epoch 0.
[I 2021-12-02 21:53:35,847] Trial 99 pruned. Trial was pruned at epoch 0.

Study statistics:
  Number of finished trials:  100
  Number of pruned trials:  67
  Number of complete trials:  33
Best trial:
  Value:  0.6713982820510864
  Params:
    convs: 3
    pooling: avg
    flatten: global
    lr: 0.0013234805544161227
```

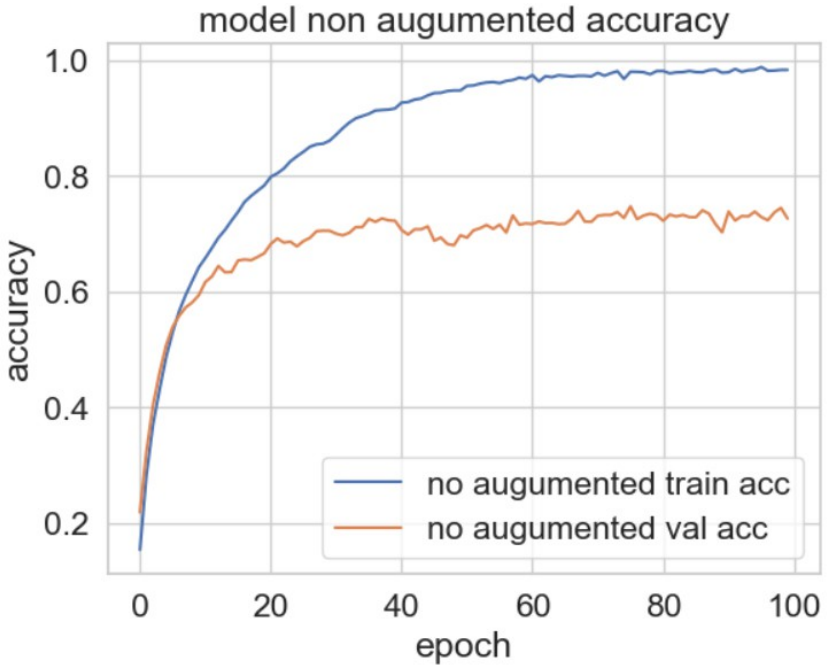
Augumentacja – gdy mamy za mało danych

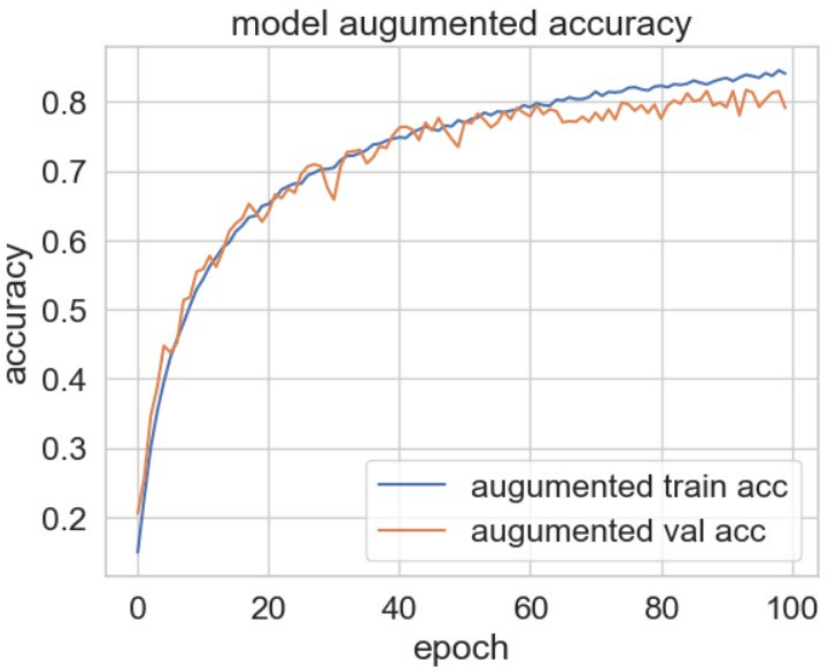
bez augumentacji

z augumentacją


```
In [25]: data_augmentation = tf.keras.Sequential([tf.keras.layers.RandomContrast(factor=0.2),
                                                 tf.keras.layers.RandomRotation(factor=0.1),
                                                 tf.keras.layers.RandomFlip('horizontal'),
                                                 tf.keras.layers.RandomFlip('vertical'),
                                                 tf.keras.layers.RandomTranslation(height_factor=0.2, width_factor=0.2),
                                                 tf.keras.layers.RandomZoom(height_factor=0.5, width_factor=0.2)])
```

Augumentacja - wyniki

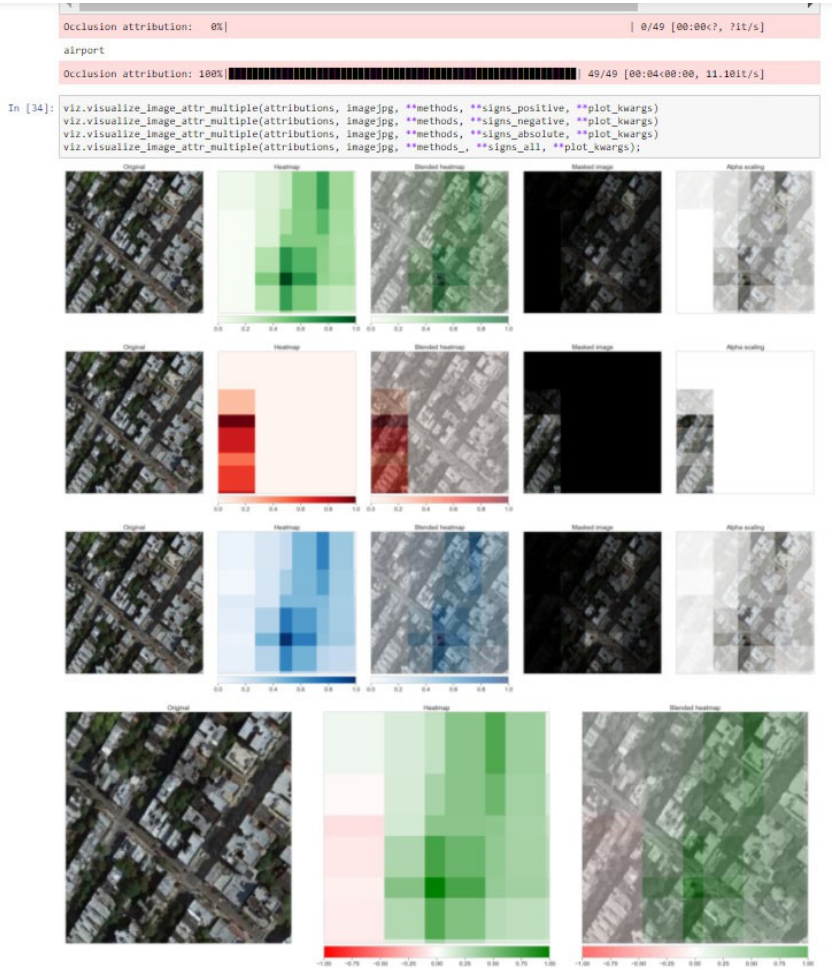




Klasyfikacja obrazu – analiza modelu (pretrained model resnet50)

- Occlusion sensitivity – klasa właściwa (dense residental)

- Occlusion sensitivity – klasa niewłaściwa (airport)

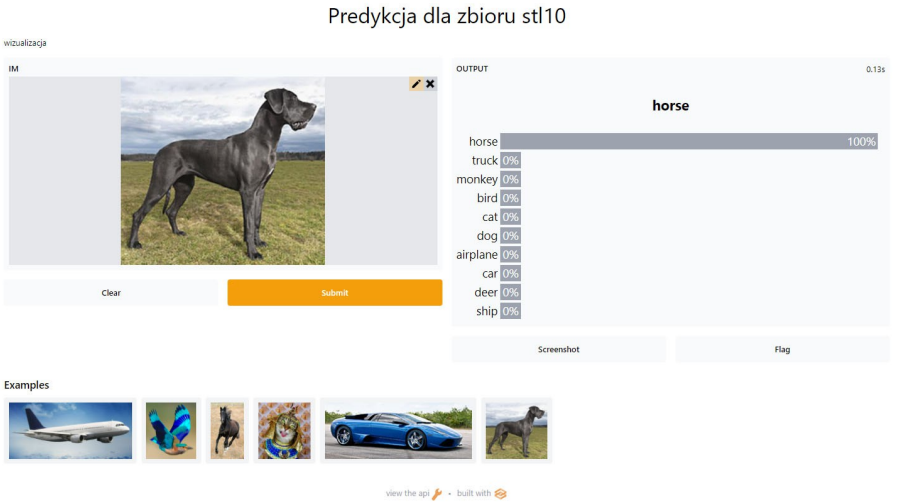


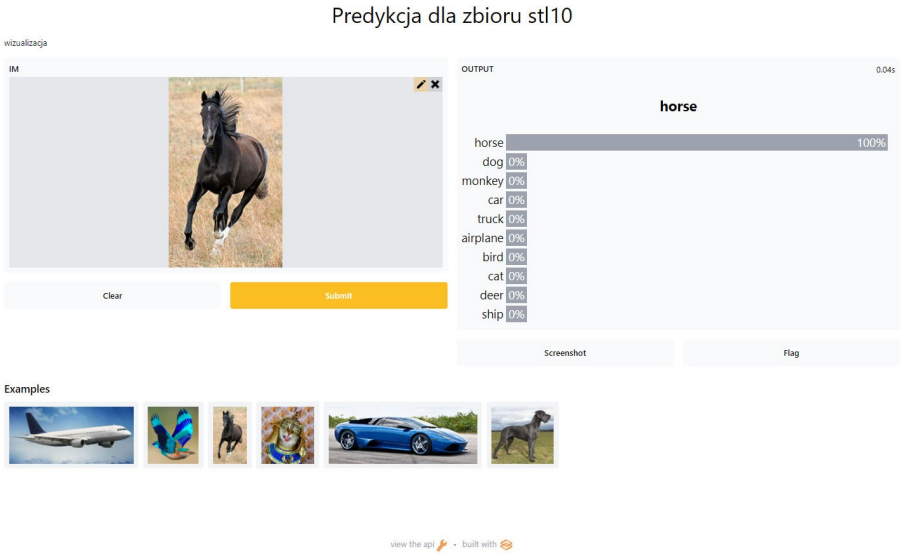
Klasyfikacja obrazu – analiza modelu (pretrained model resnet50)

- (Guided) GradCAM – klasa właściwa (dense residental)

- (Guided) GradCAM – klasa niewłaściwa (golf course)

Klasyfikacja obrazu - Niezbyt udane wdrożenie jako aplikacja demo czyli jak oszukać AI (gradio) ;)





Gradio kontra pretrenowane modele ;)