Aim

Using the code in the lesson to classify the MNIST dataset, improve the performance of the perceptron-based model by changing the ***dropout***, ***activation function***, and ***optimization algorithm***. Make a report on model tuning, give a combination of parameters that provide the best quality indicators.

Approach

Having a code from the lesson, firstly, two functions were created, so we could use our model more easily: ‘create_model’, which creates and returns an instance of a neural network (n.n.), ‘train_model’, which adjusts weights and tests the results.

```
def create_model(activation_function, dropout: float, optimizer: str):
    feature_vector_length = 784
    num_classes = 10
    input_shape = (feature_vector_length,)

    model = Sequential()
    model.add(Dense(350, input_shape=input_shape, activation=activation_function))
    model.add(Dropout(dropout))
    model.add(Dense(50, activation=activation_function))
    model.add(Dropout(dropout))
    model.add(Dense(num_classes, activation='softmax'))  # last layer
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

```
def train_model(model, batch_size, learning_rate, num_epochs):
    log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tensorflow.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
    model.fit(X_train, Y_train, epochs=num_epochs, batch_size=batch_size, verbose=0, validation_split=learning_rate,
              callbacks=[tensorboard_callback])
    return None
```

Optuna framework can help in choosing the best hyperparameters - is an optimizator, which will maximize our result (accuracy) by changing those parameters. According to documentation, the code should be written as following:

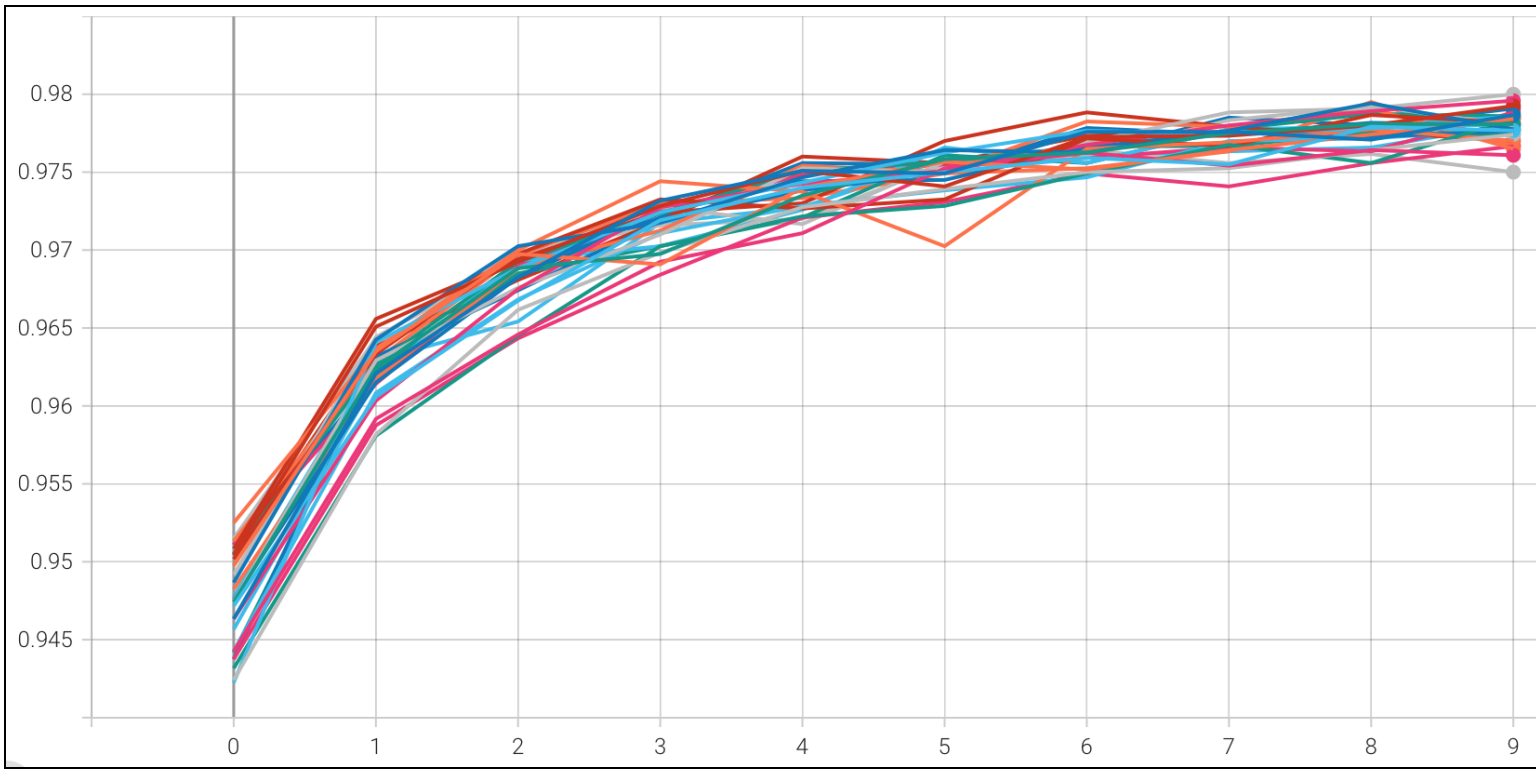
```python
def objective(trial):
    params = {
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-5, 1e-1),
        'optimizer': trial.suggest_categorical("optimizer", ["Adam", "RMSprop", "SGD"]),
        'num_epochs': trial.suggest_int('num_epochs', 3, 15),
        'batch_size': trial.suggest_int('batch_size', 20, 300),
        'activation_function': trial.suggest_categorical('activation_function', activation_functions),
        'dropout': trial.suggest_float('dropout', 0.1, 0.9)

    }
```

```python
    model = create_model(
        # optimizer=params['optimizer'],
        optimizer='RMSprop',
        # activation_function=params['activation_function'],
        activation_function='relu',
        # dropout=params['dropout']
        dropout=0.24
    )
    train_model(
        model=model,
        # learning_rate=params['learning_rate'],
        learning_rate=0.001,
        # batch_size=params['batch_size'],
        batch_size=250,
        # num_epochs=params['num_epochs']
        num_epochs=10
    )

    # accuracy = train_and_evaluate(params, model)
    test_results = model.evaluate(X_test, Y_test, verbose=0)
    accuracy = test_results[1]
    return accuracy


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30)
```

All the results will be logged via TensorBoard.

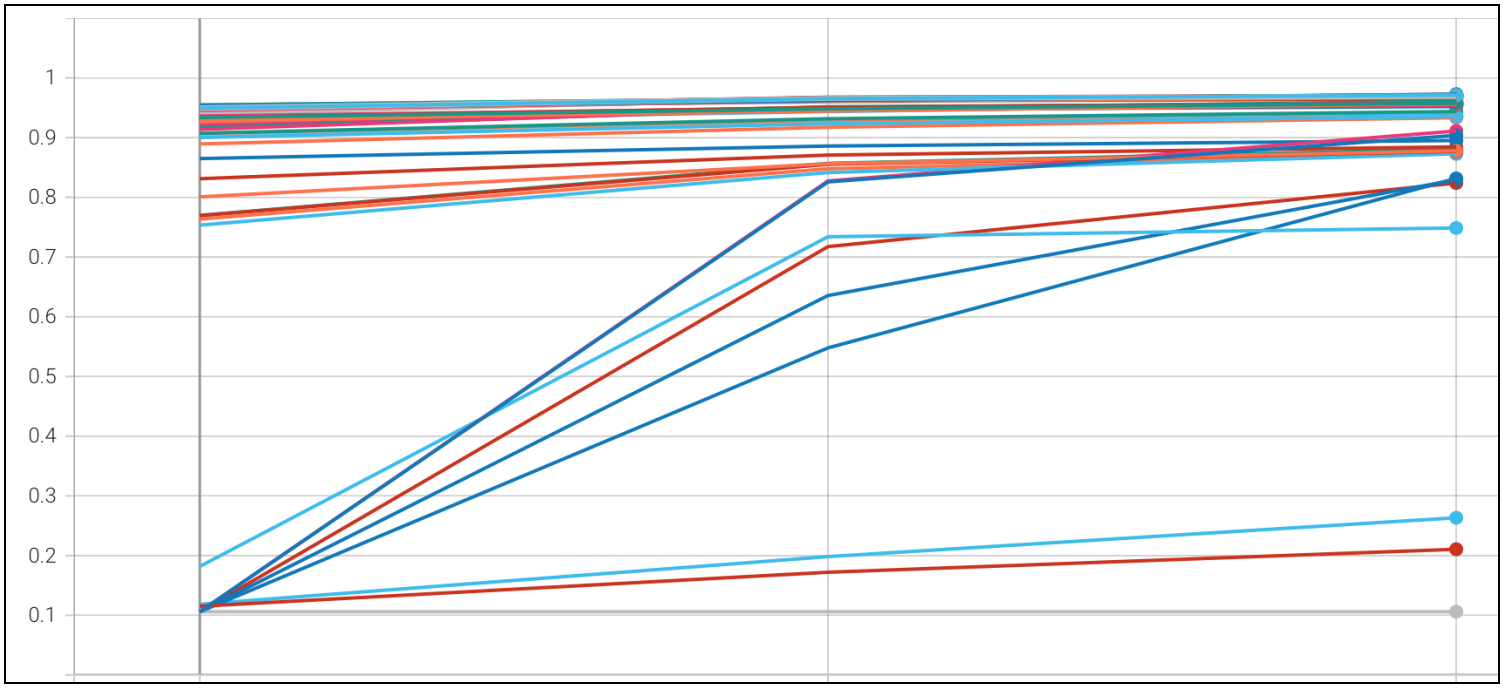
Tests

Let’s start with locking our activation function (a.f.) and optimizer (‘relu’, ‘Adam’), to find the most suitable dropout coefficient between our layers. Number of Optuna trials is 30. In our model we have 10 epochs, 0.2 as a learning rate and batch size of 250.



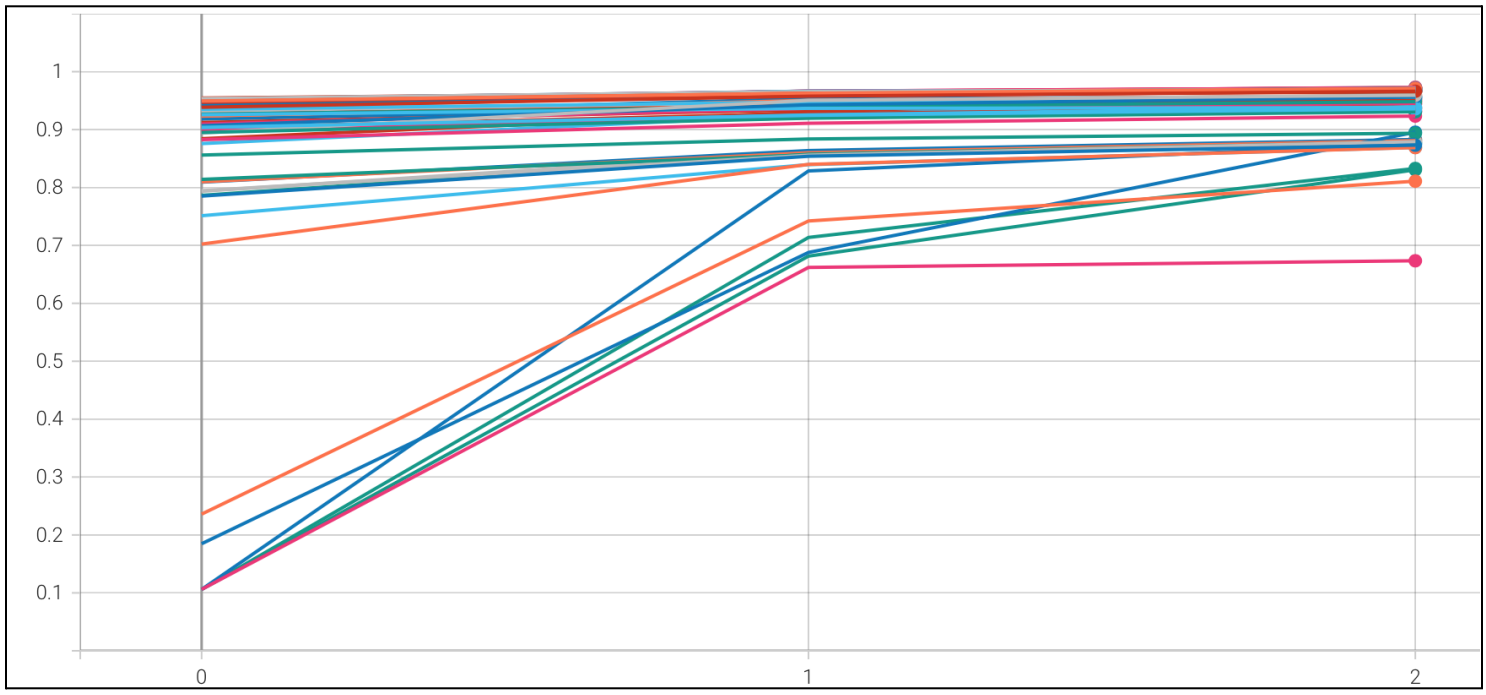
On this graph we can observe some trials of Optuna to find the dropout variable. So the best performance of ~0.9814 was achieved with the hyperparameter of ~0.1. We also can notice that all the attempts can be described with the same pattern of growth, but some anomalies are present.

Next, let’s find out which activation function and optimizator will do their best for our number recognition n.n. We have a choice of "Adam", "RMSprop", "SGD" as optimizators and 'relu', 'sigmoid', 'softmax', 'tanh', 'selu', 'exponential' as activation functions. In this experiment, the number of epochs will be reduced to 3, so time will be saved for the more trials of Optuna - 100 .



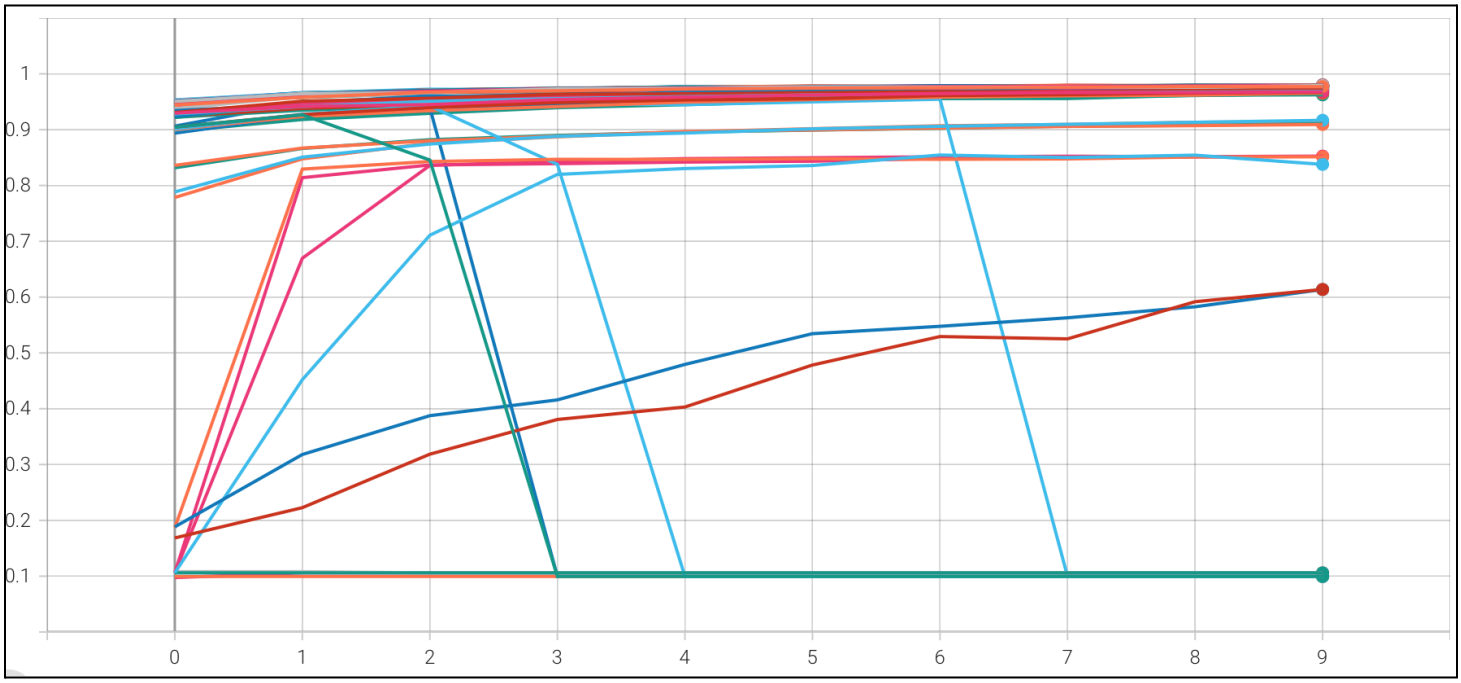
After 100 trials, we definitely can say that the best pair of parameters are ‘relu’ and RMSprop. Looking at the numerical report of the experiment, it’s noticeable that exactly those 2 did the best in comparison with other combinations. On trial 89 our neural network performed with a best result of ~0.9743.

Not less important, to try determine all the 3 parameters together.



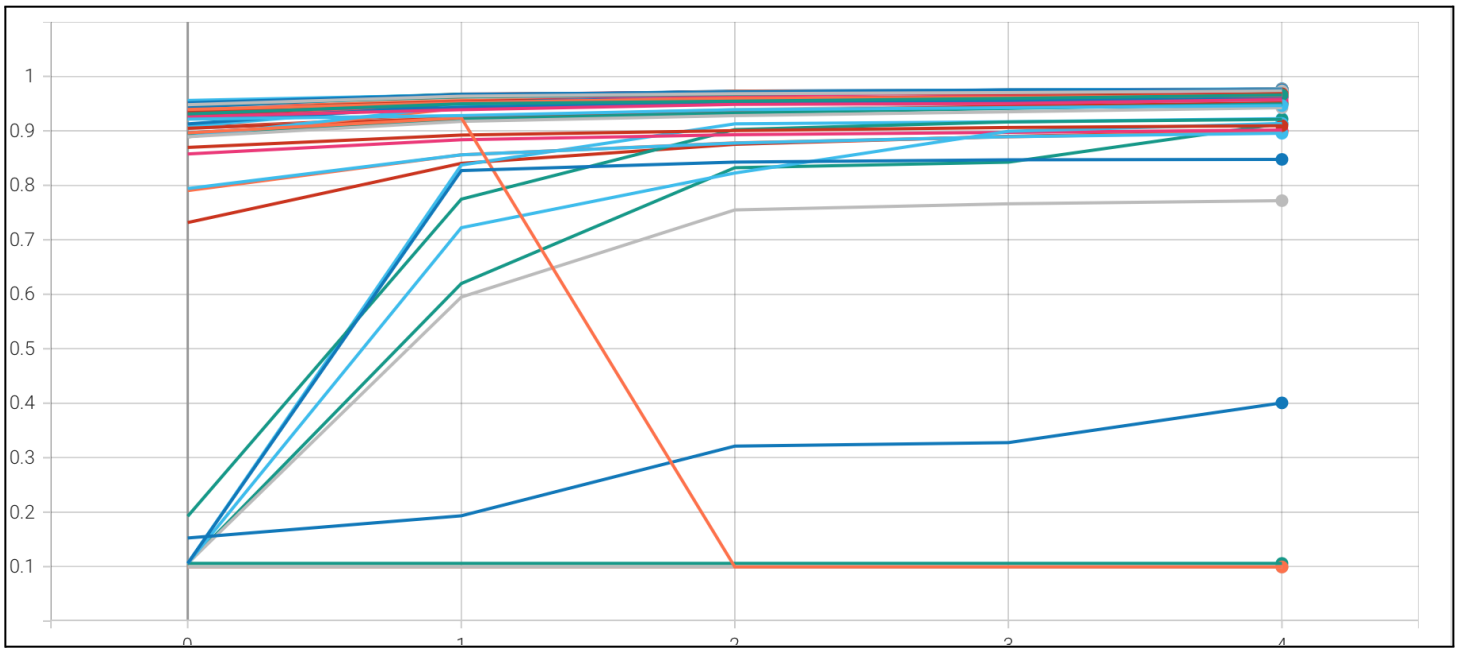
Surprisingly, the a.f. and optimizer remained the same, but dropout changed to 0.009.

If we perform the same experiment with an increased number of epochs (10), the dropout parameter will change to ~0.179 with performance of ~0.981. We can state that its value definitely depends on the number of epochs.



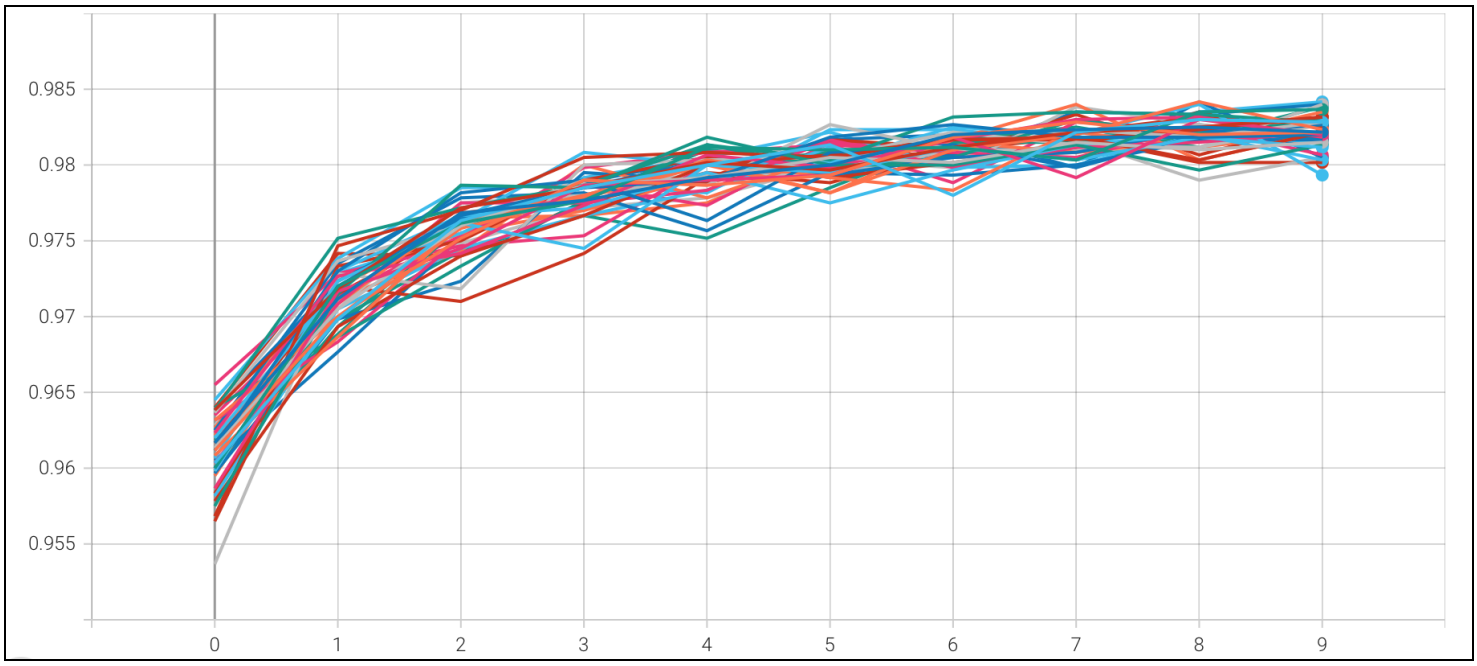
After decreasing the number of epochs to 5, the pattern can be noticed:

10 epochs - ~0.179, 5 epochs - ~0.077, 3 epochs - ~0.009. It reduces with a decrease of epochs number.



Final evaluation

This picture shows us the graphical result of 50 trainings of our model with found parameters. On average the performance was ~0.983 in the 10th epoch. The highest and lowest are ~0.9845 and ~0.979, which is quite good performance.



Conclusion

To sum up, we can say that every single hyperparameter of our number recognition neural network created with the help of TensorFlow framework is very sensitive to little change of other variables. It's not a trivial aim to stabilize the behavior of our model. A set of final accurancies will always be distributed around some mean. We've used the Optuna tool to automate the process of determining parameters, it was done well. With found variables, our n.n. achieved an accuracy of ~0.983 on average.