РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

Дисциплина: Методы машинного обучения

Студент: Кузнецов Юрий Владимирович

Группа: НФИбд 01-20

Москва 2023

Вариант №21

1. Набор данных deep_weeds
2. Классы с метками 1,2,3

Задание:

В соответствии с индивидуальным заданием, указанным в записной книжке команды, выполните следующие работы:

1. Загрузите заданный в индивидуальном задании набор данных с изображениями из Tensorflow Datasets с разбиением на обучающую и тестовую выборки.
2. Визуализируйте несколько изображений, отобранных случайным образом из обучающей выборки.
3. Оставьте в наборе изображения двух классов, указанных в индивидуальном задании первыми. Обучите нейронные сети MLP и CNN задаче бинарной классификации изображений (архитектура сетей по вашему усмотрению).
4. Постройте кривые обучения нейронных сетей для показателей ошибки и аккуратности в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.
5. Сравните качество бинарной классификации нейронными сетями при помощи матрицы ошибок для тестовой выборки.
6. Визуализируйте ROC-кривые для построенных классификаторов на одном рисунке (с легендой) и вычислите площади под ROC-кривыми.
7. Оставьте в наборе изображения трех классов, указанных в индивидуальном задании. Обучите нейронные сети MLP и CNN задаче многоклассовой классификации изображений (архитектура сетей по вашему усмотрению).
8. Сравните качество многоклассовой классификации нейронными сетями при помощи матрицы ошибок (для трех классов) для тестовой выборки.
9. Постройте кривые обучения нейронных сетей для показателей ошибки и аккуратности в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

▾ Решение

```
# !pip install -q tfds-nightly
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import tensorflow_datasets as tfds
from PIL import Image, ImageOps
```

1. Загрузим заданный в индивидуальном задании набор данных с изображениями из Tensorflow Datasets с разбиением на обучающую и тестовую выборки:

```
ds_train = tfds.load("deep_weeds", split='train[:50%]')
ds_test = tfds.load("deep_weeds", split='train[50%:57%]')
df_train = tfds.as_dataframe(ds_train)
df_test  = tfds.as_dataframe(ds_test)
df_train.shape, df_test.shape
```

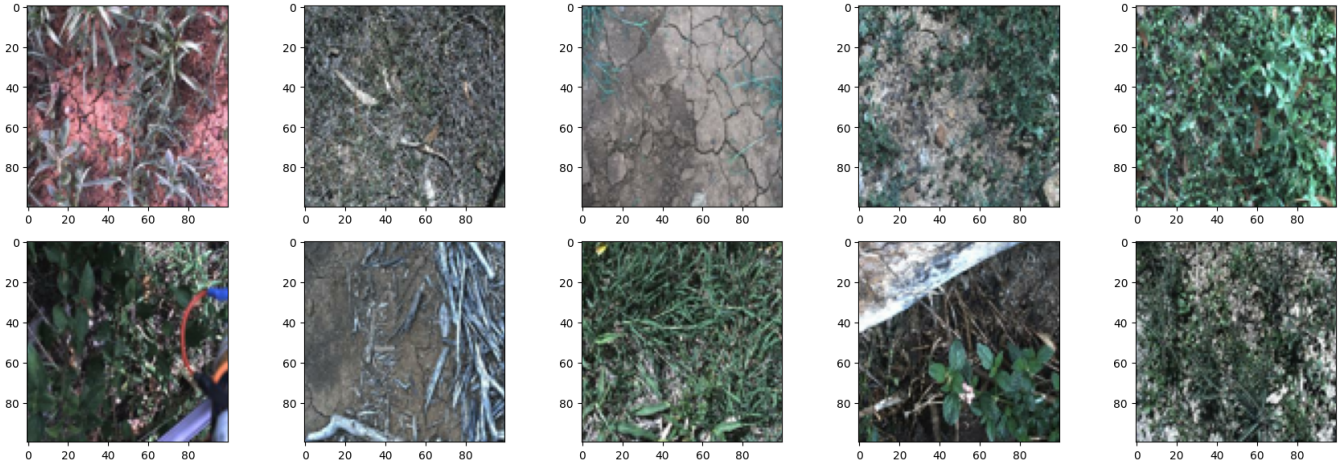
```
    ((8754, 2), (1226, 2))
```

```
def convert_to_labels_and_images(df_train, df_test):
  train_labels = df_train['label'].to_numpy(dtype=np.float32)
  test_labels = df_test['label'].to_numpy(dtype=np.float32)

  train_images = np.zeros(shape=(df_train.shape[0],100, 100, 3), dtype=np.float3
  test_images  = np.zeros(shape=(df_test.shape[0],100, 100, 3), dtype=np.float32

  for idx in range(train_labels.shape[0]):
    train_images[idx,:,:,:] = np.array(Image.fromarray(df_train.iloc[idx]['image

  for idx in range(test_labels.shape[0]):
    test_images[idx,:,:,:] = np.array(Image.fromarray(df_test.iloc[idx]['image']

  train_images /= 255
  test_images  /= 255

  return train_images, train_labels, test_images, test_labels


train_images, train_labels, test_images, test_labels = convert_to_labels_and_ima
```

2. Визуализируем несколько изображений, отобранных случайным образом из обучающей выборки:

```
import random

def plot_random_sample(images):
    n = 10
    imgs = random.sample(list(images), n)

    num_row = 2
    num_col = 5

    fig, axes = plt.subplots(num_row, num_col, figsize=(3.5 * num_col, 3 * num_r
    # For every image
    for i in range(num_row * num_col):
        # Read the image
        img = imgs[i]
        # Display the image
        ax = axes[i // num_col, i % num_col]
        ax.imshow(img)

    plt.tight_layout()
    plt.show()
```

```
plot_random_sample(train_images)
```



3. Оставим в наборе только изображения классов 1 и 2 :

```
df_train = tfds.as_dataframe(ds_train)
df_test  = tfds.as_dataframe(ds_test)

df_train = df_train.loc[((df_train['label'] == 1) | (df_train['label' ] == 2))]
df_test = df_test.loc[((df_test['label'] == 1) | (df_test['label' ] == 2))]


df_train.shape, df_test.shape
```

```
    ((1053, 2), (149, 2))
```

```
train_images, train_labels, test_images, test_labels = convert_to_labels_and_ima
train_labels -= 1 #  вычитаем 1, чтобы
test_labels -= 1 #  классы обозначались как 0 и 1
```

Обучим нейронную сеть MLP задаче бинарной классификации изображений:

```
tf.random.set_seed(42)

model_1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(100, 100, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_1.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)

history_1 = model_1.fit(
    train_images,
    train_labels,
    epochs=50,
    batch_size=32,
    validation_data=(test_images, test_labels)
)
```

```
Epoch 1/50
33/33 [==============================] - 6s 77ms/step - loss: 1.2157 - accu
Epoch 2/50
33/33 [==============================] - 2s 59ms/step - loss: 0.6700 - accu
Epoch 3/50
33/33 [==============================] - 2s 51ms/step - loss: 0.6041 - accu
Epoch 4/50
33/33 [==============================] - 2s 62ms/step - loss: 0.5429 - accu
Epoch 5/50
33/33 [==============================] - 2s 50ms/step - loss: 0.5485 - accu
Epoch 6/50
33/33 [==============================] - 3s 76ms/step - loss: 0.5248 - accu
Epoch 7/50
33/33 [==============================] - 2s 55ms/step - loss: 0.5966 - accu
Epoch 8/50
33/33 [==============================] - 2s 49ms/step - loss: 0.3863 - accu
Epoch 9/50
33/33 [==============================] - 2s 48ms/step - loss: 0.3101 - accu
Epoch 10/50
33/33 [==============================] - 2s 48ms/step - loss: 0.3614 - accu
Epoch 11/50
33/33 [==============================] - 2s 49ms/step - loss: 0.4124 - accu
Epoch 12/50
```

```
33/33 [==============================] - 2s 49ms/step - loss: 0.2167 - accu
Epoch 13/50
33/33 [==============================] - 2s 59ms/step - loss: 0.1512 - accu
Epoch 14/50
33/33 [==============================] - 2s 65ms/step - loss: 0.1306 - accu
Epoch 15/50
33/33 [==============================] - 2s 47ms/step - loss: 0.0879 - accu
Epoch 16/50
33/33 [==============================] - 2s 48ms/step - loss: 0.1421 - accu
Epoch 17/50
33/33 [==============================] - 2s 48ms/step - loss: 0.0829 - accu
Epoch 18/50
33/33 [==============================] - 2s 63ms/step - loss: 0.0438 - accu
Epoch 19/50
33/33 [==============================] - 2s 58ms/step - loss: 0.0390 - accu
Epoch 20/50
33/33 [==============================] - 2s 58ms/step - loss: 0.0249 - accu
Epoch 21/50
33/33 [==============================] - 2s 66ms/step - loss: 0.0225 - accu
Epoch 22/50
33/33 [==============================] - 2s 48ms/step - loss: 0.0147 - accu
Epoch 23/50
33/33 [==============================] - 2s 49ms/step - loss: 0.0120 - accu
Epoch 24/50
33/33 [==============================] - 2s 49ms/step - loss: 0.0107 - accu
Epoch 25/50
33/33 [==============================] - 2s 48ms/step - loss: 0.0077 - accu
Epoch 26/50
33/33 [==============================] - 2s 49ms/step - loss: 0.0071 - accu
Epoch 27/50
33/33 [==============================] - 2s 49ms/step - loss: 0.0055 - accu
Epoch 28/50
33/33 [==============================] - 2s 74ms/step - loss: 0.0049 - accu
Epoch 29/50
33/33 [==============================] - 2s 54ms/step - loss: 0.0037 - accu
Epoch 30/50
33/33 [==============================] - 2s 48ms/step - loss: 0.0033 - accu
```

Обучим нейронную сеть CNN задаче бинарной классификации изображений:

```
tf.random.set_seed(42)

model_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), input_shape=(100, 100
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(128, activation='relu'),
```

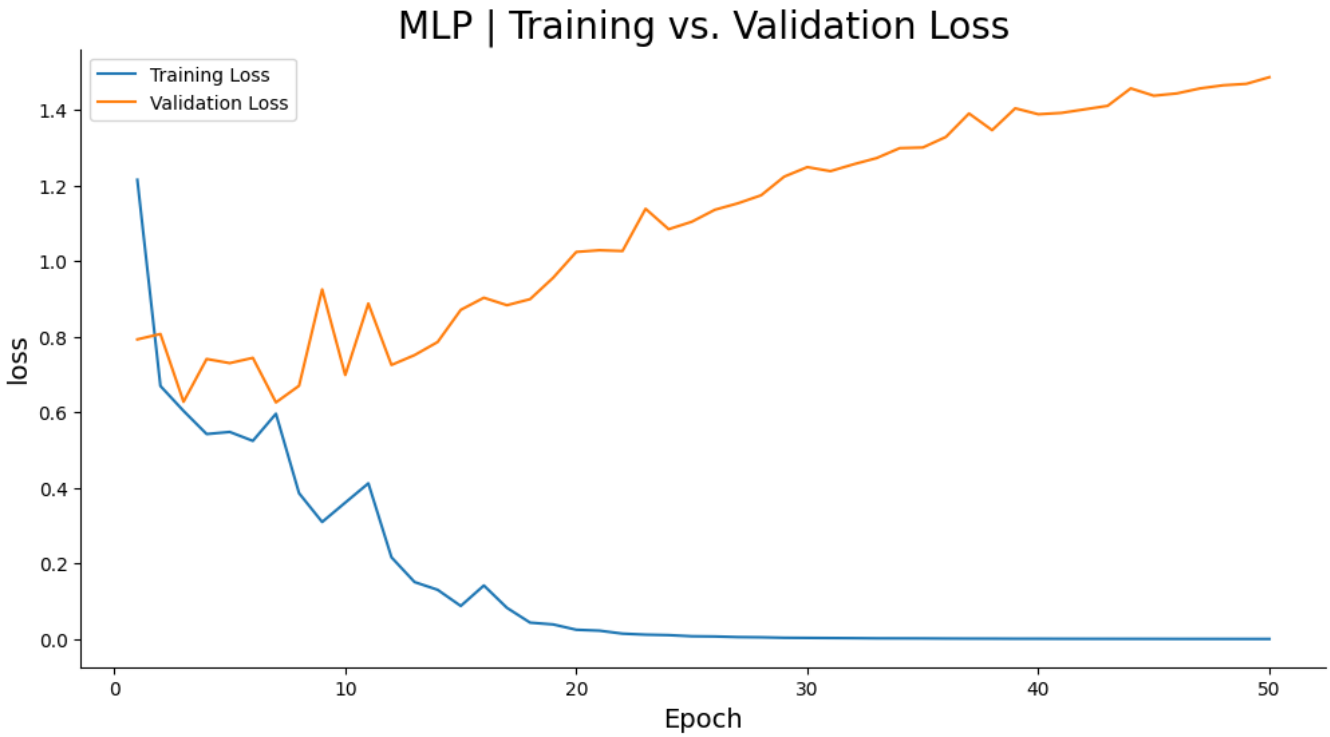
```
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_2.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)

history_2 = model_2.fit(
    train_images,
    train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(test_images, test_labels)
)
```

```
Epoch 1/15
33/33 [==============================] - 12s 316ms/step - loss: 0.6236 - ac
Epoch 2/15
33/33 [==============================] - 8s 249ms/step - loss: 0.3340 - acc
Epoch 3/15
33/33 [==============================] - 10s 291ms/step - loss: 0.2061 - ac
Epoch 4/15
33/33 [==============================] - 10s 314ms/step - loss: 0.1260 - ac
Epoch 5/15
33/33 [==============================] - 12s 377ms/step - loss: 0.0999 - ac
Epoch 6/15
33/33 [==============================] - 9s 277ms/step - loss: 0.1180 - acc
Epoch 7/15
33/33 [==============================] - 10s 304ms/step - loss: 0.0429 - ac
Epoch 8/15
33/33 [==============================] - 11s 348ms/step - loss: 0.0369 - ac
Epoch 9/15
33/33 [==============================] - 14s 443ms/step - loss: 0.0292 - ac
Epoch 10/15
33/33 [==============================] - 12s 365ms/step - loss: 0.0370 - ac
Epoch 11/15
33/33 [==============================] - 10s 311ms/step - loss: 0.0443 - ac
Epoch 12/15
33/33 [==============================] - 9s 271ms/step - loss: 0.0593 - acc
Epoch 13/15
33/33 [==============================] - 10s 289ms/step - loss: 0.0202 - ac
Epoch 14/15
33/33 [==============================] - 10s 302ms/step - loss: 0.0086 - ac
Epoch 15/15
33/33 [==============================] - 12s 358ms/step - loss: 0.0070 - ac
```

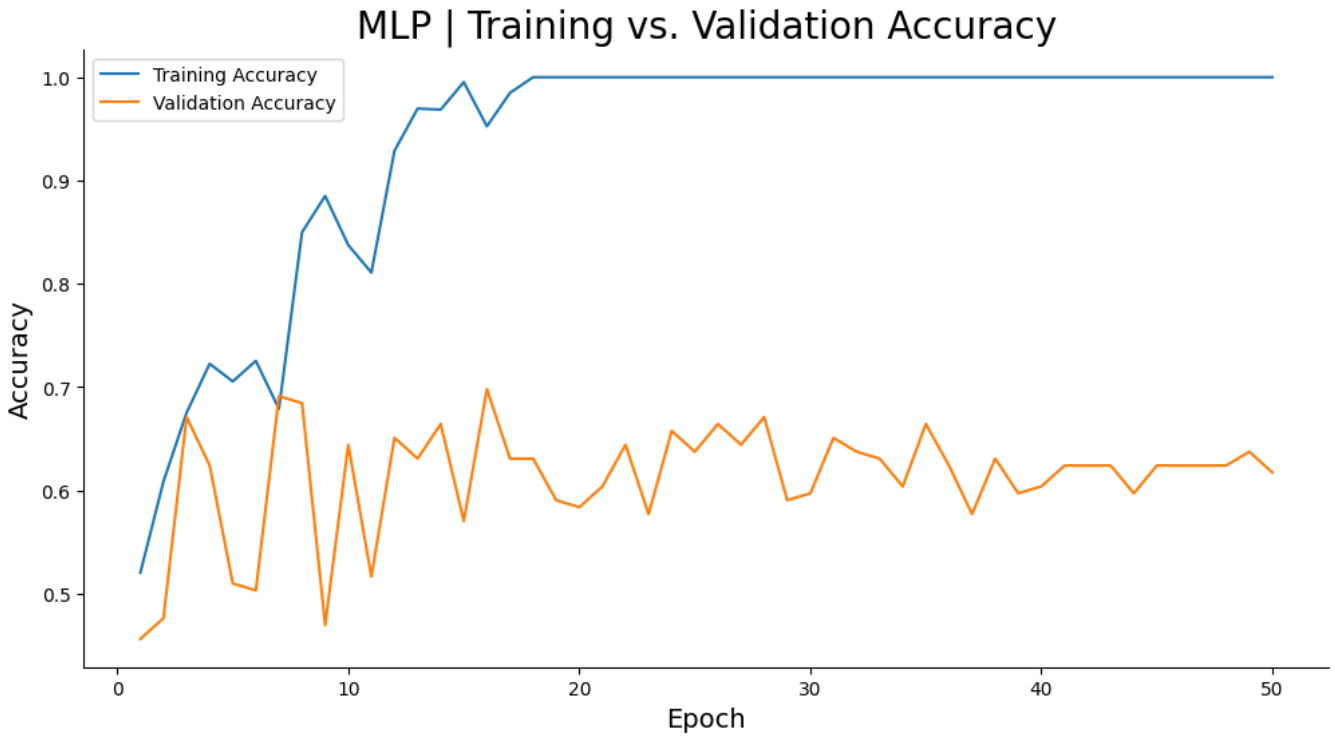
4. Построим кривые обучения нейронных сетей бинарной классификации для показателей ошибки и доли верных ответов в зависимости от эпохи обучения:

```
from matplotlib import rcParams

rcParams['figure.figsize'] = (12, 6)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

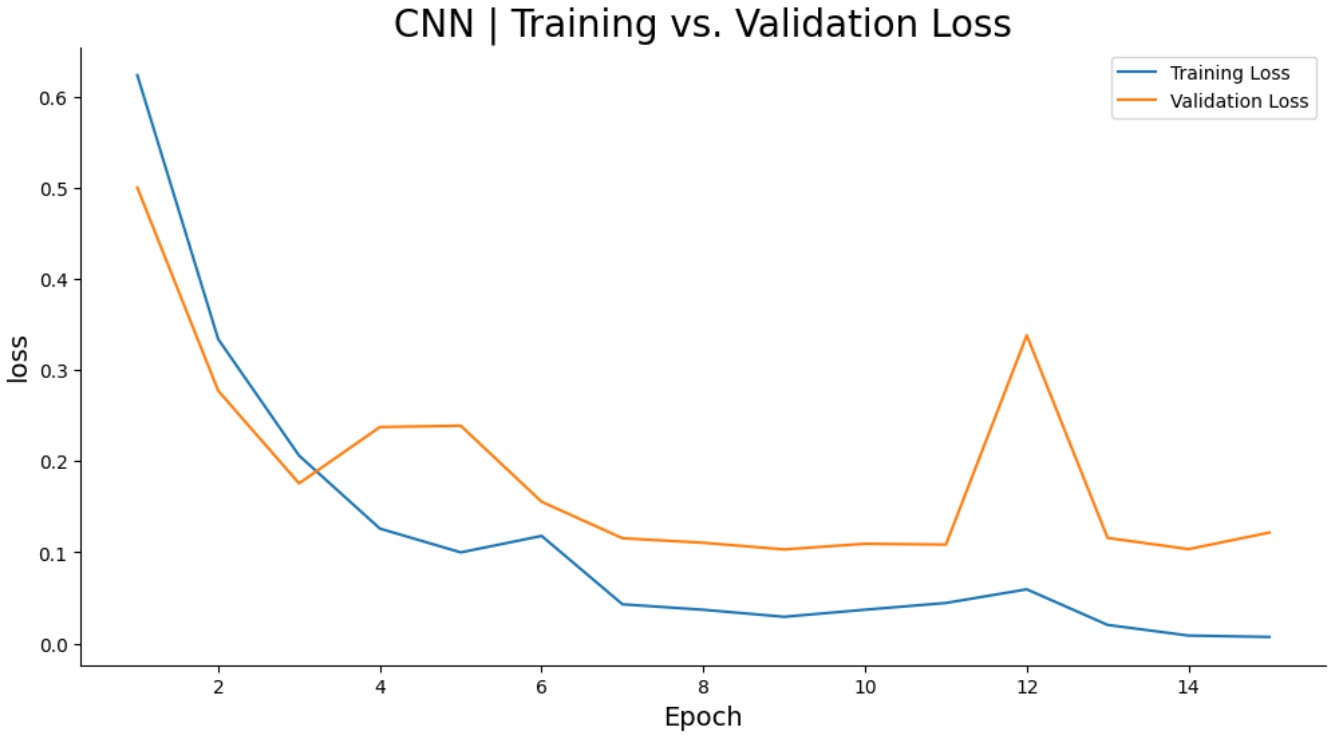
```
plt.plot(np.arange(1, 51), history_1.history['loss'], label='Training Loss')
plt.plot(np.arange(1, 51), history_1.history['val_loss'], label='Validation Loss
plt.title('MLP | Training vs. Validation Loss', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('loss', size=14)
plt.legend();
```



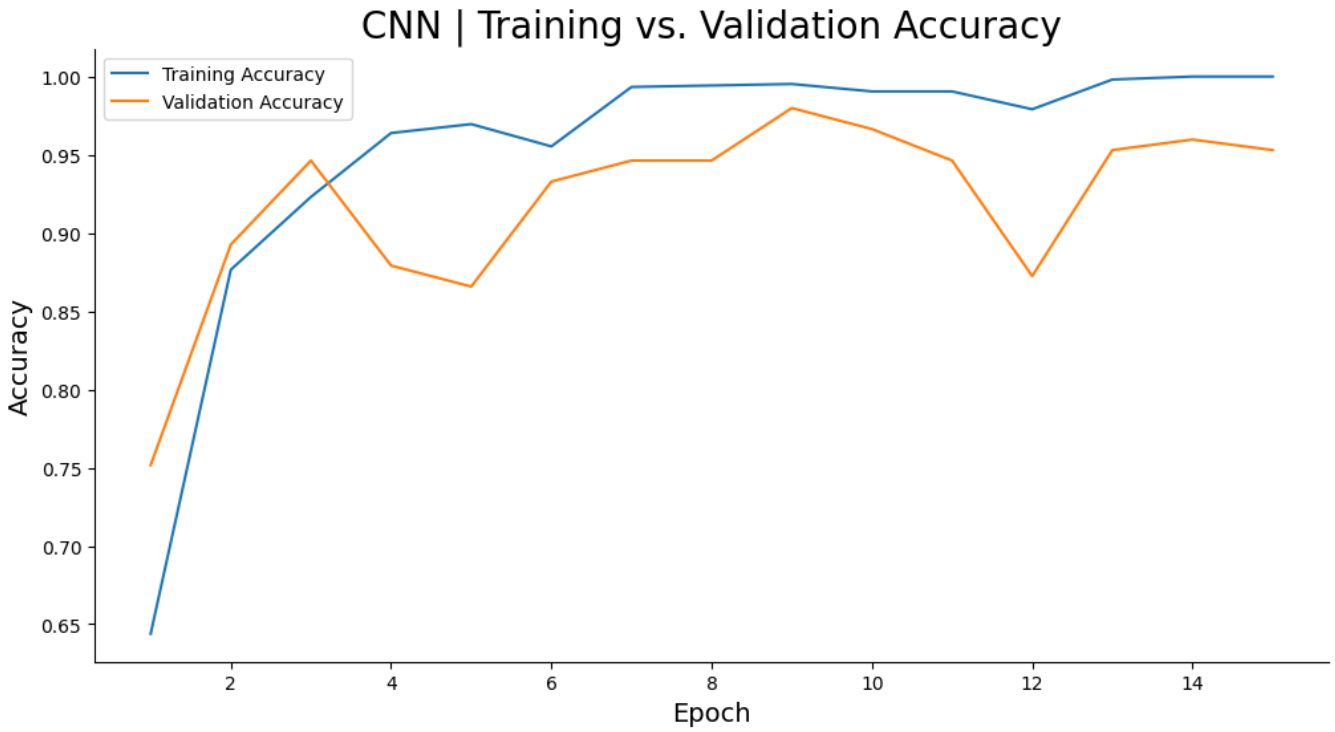
```
plt.plot(np.arange(1, 51), history_1.history['accuracy'], label='Training Accura
plt.plot(np.arange(1, 51), history_1.history['val_accuracy'], label='Validation
plt.title('MLP | Training vs. Validation Accuracy', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('Accuracy', size=14)
plt.legend();
```



```
plt.plot(np.arange(1, 16), history_2.history['loss'], label='Training Loss')
plt.plot(np.arange(1, 16), history_2.history['val_loss'], label='Validation Loss
plt.title('CNN | Training vs. Validation Loss', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('loss', size=14)
plt.legend();
```



```
plt.plot(np.arange(1, 16), history_2.history['accuracy'], label='Training Accura
plt.plot(np.arange(1, 16), history_2.history['val_accuracy'], label='Validation
plt.title('CNN | Training vs. Validation Accuracy', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('Accuracy', size=14)
plt.legend();
```



5. Сравним качество бинарной классификации нейронными сетями при помощи матрицы ошибок для тестовой выборки:

```
def TN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 0))
```

```
def FP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 1))


def FN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 0))


def TP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 1))


def confusion_matrix(y_true, y_predict):
    return np.array([
        [TP(y_true, y_predict), FN(y_true, y_predict)],
        [FP(y_true, y_predict), TN(y_true, y_predict)]
    ])
```

Вычислим матрицу ошибок для MLP:

```
prediction_1 = model_1.predict(test_images)
pred_labels_1 = np.array([1 if prob > 0.5 else 0 for prob in np.ravel(prediction
confusion_matrix(test_labels, pred_labels_1)
```

```
5/5 [==============================] - 1s 27ms/step
array([[49, 30],
       [27, 43]])
```

Вычислим матрицу ошибок для CNN:

```
prediction_2 = model_2.predict(test_images)
pred_labels_2 = np.array([1 if prob > 0.5 else 0 for prob in np.ravel(prediction
confusion_matrix(test_labels, pred_labels_2)
```

```
5/5 [==============================] - 1s 175ms/step
array([[74,  5],
       [ 2, 68]])
```

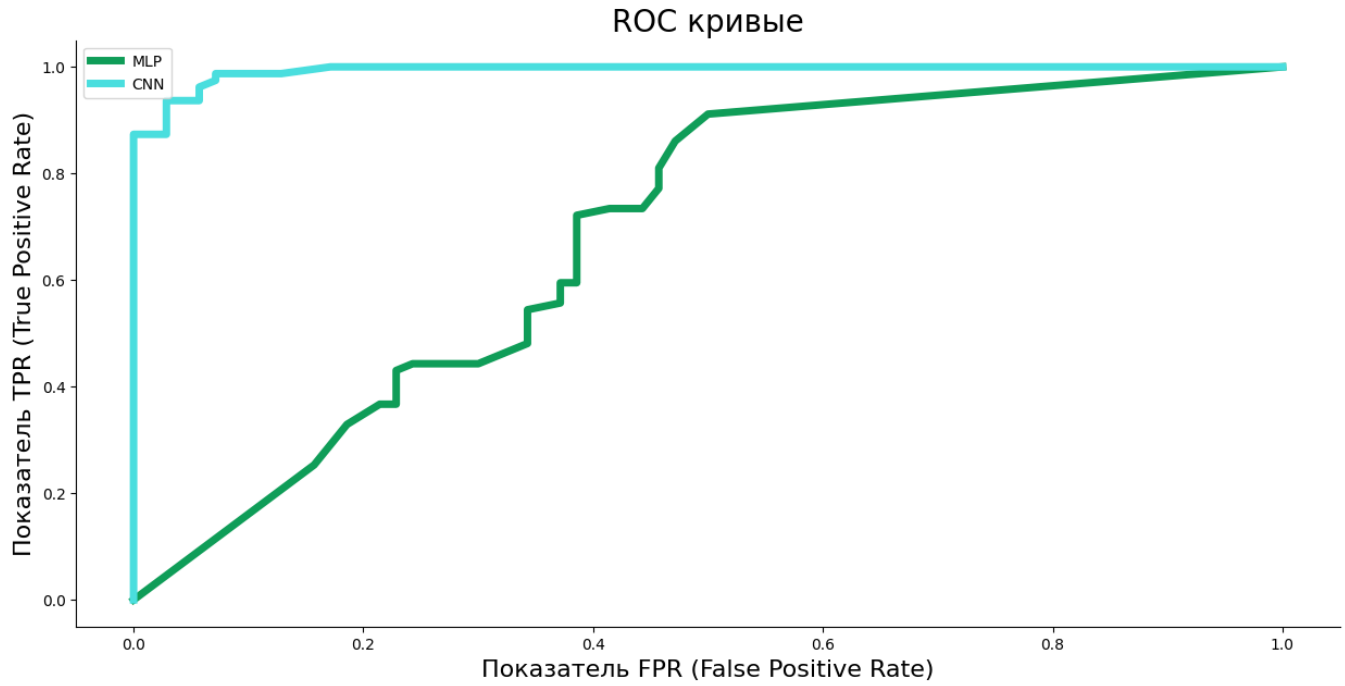
Видим, что качество бинарной классификации CNN несколько выше.

6. Визуализируем ROC-кривые для построенных классификаторов:

```
def true_false_positive(threshold_vector, y_test):
    true_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 1)
    true_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 0)
    false_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 0)
    false_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 1)

    tpr = true_positive.sum() / (true_positive.sum() + false_negative.sum())
    fpr = false_positive.sum() / (false_positive.sum() + true_negative.sum())

    return tpr, fpr


def roc_from_scratch(probabilities, y_test, partitions=100):
    roc = np.array([])
    for i in range(partitions + 1):

        threshold_vector = np.greater_equal(probabilities, i / partitions).astyp
        tpr, fpr = true_false_positive(threshold_vector, y_test)
        roc = np.append(roc, [fpr, tpr])

    return roc.reshape(-1, 2)
```

```
plt.figure(figsize=(15,7))

ROC_1 = roc_from_scratch(prediction_1.reshape(-1),test_labels,partitions=50)
ROC_2 = roc_from_scratch(prediction_2.reshape(-1),test_labels,partitions=50)
plt.plot(ROC_1[:,0],ROC_1[:,1],color='#0F9D58',lw=5)
plt.plot(ROC_2[:,0],ROC_2[:,1],color='#4ADEDE',lw=5)
plt.title('ROC кривые',fontsize=20)
plt.xlabel('Показатель FPR (False Positive Rate)',fontsize=16)
plt.ylabel('Показатель TPR (True Positive Rate)',fontsize=16)
plt.legend(['MLP', 'CNN'])
```

```
<matplotlib.legend.Legend at 0x7f85dc3198a0>
```



Вычислим площади под кривыми:

```
S_1 = 0
x_, y_ = ROC_1[:,0], ROC_1[:,1]
for i in range(len(x_) - 1):
  S_1 += abs(x_[i] - x_[i+1]) * (y_[i] + y_[i+1]) / 2
S_1
```

```
0.686618444846293
```

```
S_2 = 0
x_, y_ = ROC_2[:,0], ROC_2[:,1]
for i in range(len(x_) - 1):
  S_2 += abs(x_[i] - x_[i+1]) * (y_[i] + y_[i+1]) / 2
S_2
```

```
0.9931283905967453
```

```
print("Площадь под ROC кривой для MLP: {:.2}".format(S_1))
print("Площадь под ROC кривой для CNN: {:.2}".format(S_2))
```

```
Площадь под ROC кривой для MLP: 0.69
Площадь под ROC кривой для CNN: 0.99
```

7. Оставим в наборе изображения классов 6, 7, 8

```
df_train = tfds.as_dataframe(ds_train)
df_test  = tfds.as_dataframe(ds_test)

df_train = df_train.loc[((df_train['label'] == 1) | (df_train['label' ] == 2))]
df_test = df_test.loc[((df_test['label'] == 1) | (df_test['label' ] == 2))]
```

```
df_train.shape, df_test.shape
```

```
((1053, 2), (149, 2))
```

```
train_images, train_labels, test_images, test_labels = convert_to_labels_and_ima
train_labels -= 1 #  вычитаем 6, чтобы
test_labels -= 1 #  классы обозначались как 0, 1 и 2
```

```
train_labels.shape, test_labels.shape
```

```
((1053,), (149,))
```

Подготовим данные на вход:

```
def to_one_hot(labels, dimension=3):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results


train_labels = to_one_hot(train_labels.astype('int32'))
test_labels = to_one_hot(test_labels.astype('int32'))
train_labels.shape, test_labels.shape
```

```
((1053, 3), (149, 3))
```

Обучим нейронную сеть MLP задаче многоклассовой классификации изображений:

```
tf.random.set_seed(42)

model_1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(100, 100, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model_1.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=[tf.keras.metrics.CategoricalAccuracy(name="accuracy")]
)

history_1 = model_1.fit(
    train_images,
    train_labels,
    epochs=50,
    batch_size=32,
    validation_data=(test_images, test_labels)
)
```

```
Epoch 1/50
```

```
Epoch 1/50
33/33 [==============================] - 4s 81ms/step - loss: 1.6756 - accu
Epoch 2/50
33/33 [==============================] - 3s 98ms/step - loss: 0.8176 - accu
Epoch 3/50
33/33 [==============================] - 3s 87ms/step - loss: 0.6272 - accu
Epoch 4/50
33/33 [==============================] - 2s 72ms/step - loss: 0.5935 - accu
Epoch 5/50
33/33 [==============================] - 2s 72ms/step - loss: 0.4908 - accu
Epoch 6/50
33/33 [==============================] - 2s 73ms/step - loss: 0.3937 - accu
Epoch 7/50
33/33 [==============================] - 3s 78ms/step - loss: 0.6507 - accu
Epoch 8/50
33/33 [==============================] - 3s 101ms/step - loss: 0.3472 - acc
Epoch 9/50
33/33 [==============================] - 2s 69ms/step - loss: 0.2744 - accu
Epoch 10/50
33/33 [==============================] - 3s 82ms/step - loss: 0.3243 - accu
Epoch 11/50
33/33 [==============================] - 3s 76ms/step - loss: 0.5331 - accu
Epoch 12/50
33/33 [==============================] - 3s 81ms/step - loss: 0.2266 - accu
Epoch 13/50
33/33 [==============================] - 3s 83ms/step - loss: 0.1680 - accu
Epoch 14/50
33/33 [==============================] - 2s 74ms/step - loss: 0.1623 - accu
Epoch 15/50
33/33 [==============================] - 2s 66ms/step - loss: 0.0959 - accu
Epoch 16/50
33/33 [==============================] - 2s 71ms/step - loss: 0.1803 - accu
Epoch 17/50
33/33 [==============================] - 3s 83ms/step - loss: 0.2336 - accu
Epoch 18/50
33/33 [==============================] - 3s 96ms/step - loss: 0.0695 - accu
Epoch 19/50
33/33 [==============================] - 2s 64ms/step - loss: 0.2465 - accu
Epoch 20/50
33/33 [==============================] - 2s 76ms/step - loss: 0.0642 - accu
Epoch 21/50
33/33 [==============================] - 2s 71ms/step - loss: 0.0547 - accu
Epoch 22/50
33/33 [==============================] - 3s 77ms/step - loss: 0.0343 - accu
Epoch 23/50
33/33 [==============================] - 3s 95ms/step - loss: 0.0306 - accu
Epoch 24/50
33/33 [==============================] - 2s 69ms/step - loss: 0.0253 - accu
Epoch 25/50
33/33 [==============================] - 2s 57ms/step - loss: 0.0420 - accu
Epoch 26/50
33/33 [==============================] - 2s 60ms/step - loss: 0.0235 - accu
Epoch 27/50
33/33 [==============================] - 2s 68ms/step - loss: 0.0154 - accu
```

```
Epoch 28/50
33/33 [==============================] - 2s 69ms/step - loss: 0.0144 - accu
Epoch 29/50
33/33 [==============================] - 3s 85ms/step - loss: 0.0103 - accu
Epoch 30/50
33/33 [==============================] - 2s 70ms/step - loss: 0.0093 - accu
```

Обучим нейронную сеть CNN задаче многоклассовой классификации изображений:

```
tf.random.set_seed(42)

model_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), input_shape=(100, 100
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model_2.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=[tf.keras.metrics.CategoricalAccuracy(name="accuracy")]
)

history_2 = model_2.fit(
    train_images,
    train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(test_images, test_labels)
)
```

```
Epoch 1/15
33/33 [==============================] - 17s 350ms/step - loss: 0.7023 - ac
Epoch 2/15
33/33 [==============================] - 14s 417ms/step - loss: 0.4975 - ac
Epoch 3/15
33/33 [==============================] - 11s 325ms/step - loss: 0.3134 - ac
Epoch 4/15
33/33 [==============================] - 10s 311ms/step - loss: 0.2213 - ac
Epoch 5/15
33/33 [==============================] - 11s 323ms/step - loss: 0.1152 - ac
Epoch 6/15
33/33 [==============================] - 12s 351ms/step - loss: 0.1513 - ac
Epoch 7/15
33/33 [==============================] - 11s 348ms/step - loss: 0.0645 - ac
Epoch 8/15
33/33 [==============================] - 9s 265ms/step - loss: 0.0364 - acc
Epoch 9/15
33/33 [==============================] - 13s 403ms/step - loss: 0.0499 - ac
Epoch 10/15
33/33 [==============================] - 10s 314ms/step - loss: 0.0300 - ac
Epoch 11/15
33/33 [==============================] - 10s 311ms/step - loss: 0.0463 - ac
Epoch 12/15
33/33 [==============================] - 9s 282ms/step - loss: 0.0153 - acc
Epoch 13/15
33/33 [==============================] - 11s 322ms/step - loss: 0.0141 - ac
Epoch 14/15
33/33 [==============================] - 10s 310ms/step - loss: 0.0098 - ac
Epoch 15/15
33/33 [==============================] - 10s 301ms/step - loss: 0.0065 - ac
```

8. Сравним качество многоклассовой классификации нейронными сетями при помощи матрицы ошибок для тестовой выборки:

```
def confusion_matrix_3(labels_true, labels_predict):
  matrix_0_0 = np.sum((test_labels == 2) & (labels_predict == 2)) # число коррек
  matrix_0_1 = np.sum((test_labels == 2) & (labels_predict == 1)) # число точек
  matrix_0_2 = np.sum((test_labels == 2) & (labels_predict == 0)) # число точек

  matrix_1_0 = np.sum((test_labels == 1) & (labels_predict == 2)) # число точек
  matrix_1_1 = np.sum((test_labels == 1) & (labels_predict == 1)) # число коррек
  matrix_1_2 = np.sum((test_labels == 1) & (labels_predict == 0)) # число точек

  matrix_2_1 = np.sum((test_labels == 0) & (labels_predict == 2)) # число точек
  matrix_2_2 = np.sum((test_labels == 0) & (labels_predict == 1)) # число точек
  matrix_2_0 = np.sum((test_labels == 0) & (labels_predict == 0)) # число коррек

  return np.array([
        [matrix_0_0, matrix_0_1, matrix_0_2],
        [matrix_1_0, matrix_1_1, matrix_1_2],
        [matrix_2_0, matrix_2_1, matrix_2_2]
    ])


prediction_1 = model_1.predict(test_images)
prediction_2 = model_2.predict(test_images)
```

```
5/5 [==============================] - 0s 11ms/step
5/5 [==============================] - 0s 59ms/step
```

```
test_labels = np.array([np.argmax(prob) for prob in test_labels])
pred_labels_1 = np.array([np.argmax(prob) for prob in prediction_1])
confusion_matrix_3(test_labels, pred_labels_1)
```

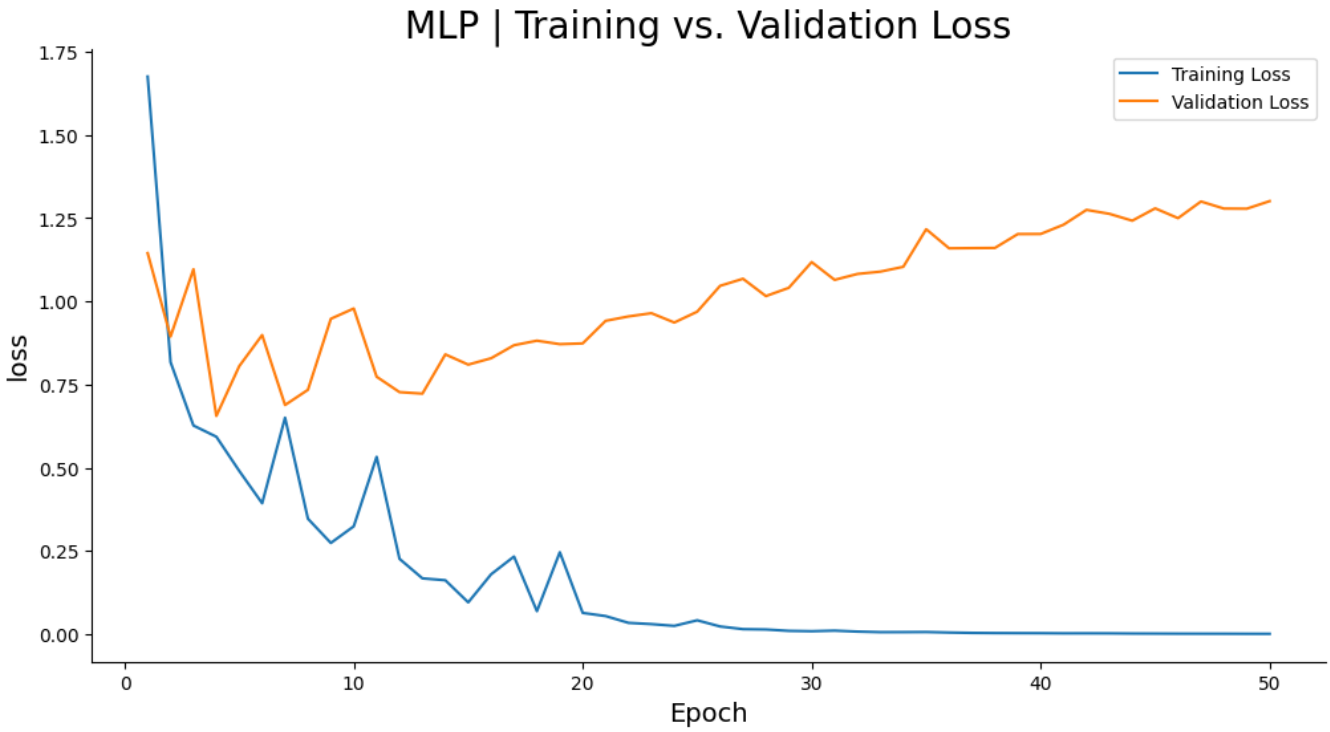
```
array([[ 0,  0,  0],
       [ 0, 48, 31],
       [44,  0, 26]])
```

```
pred_labels_2 = np.array([np.argmax(prob) for prob in prediction_2])
confusion_matrix_3(test_labels, pred_labels_2)
```

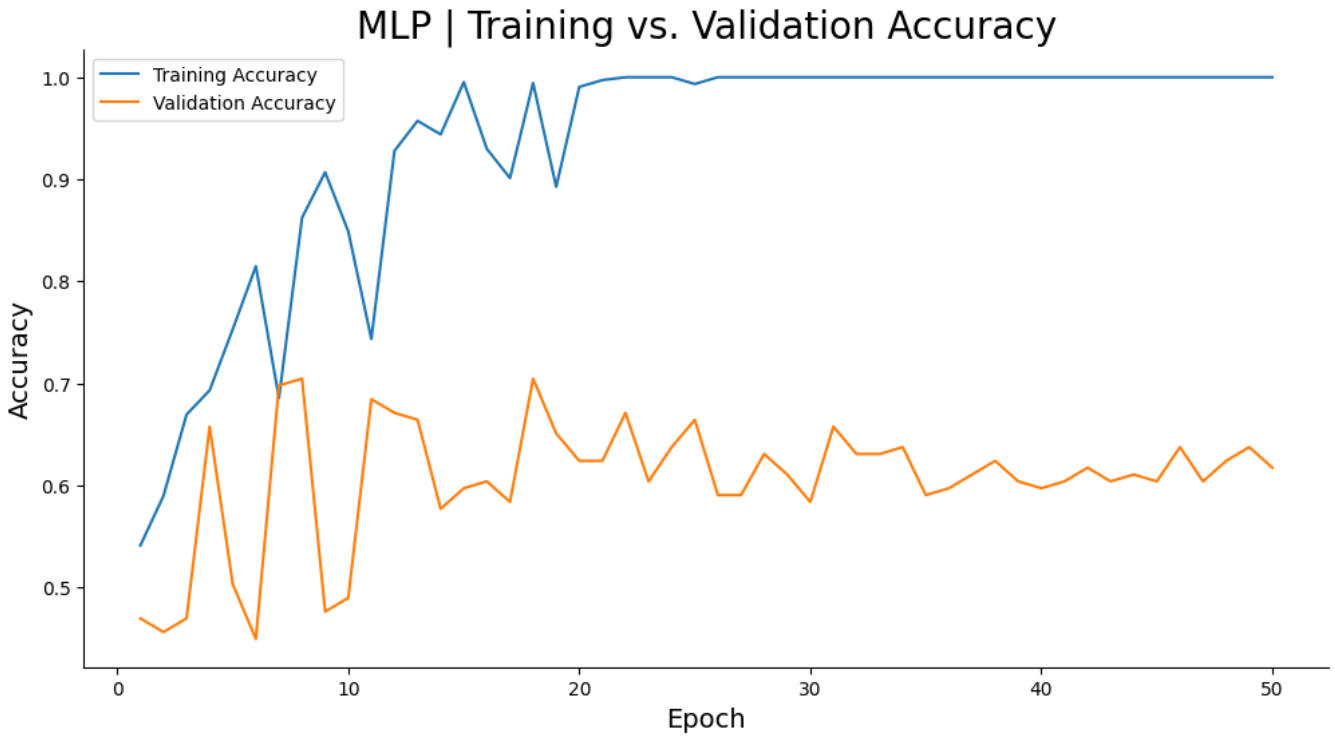
```
array([[ 0,  0,  0],
       [ 0, 77,  2],
       [57,  0, 13]])
```

9. Построиме кривые обучения нейронных сетей многоклассовой классификации для показателей ошибки и доли верных ответов в зависимости от эпохи обучения.

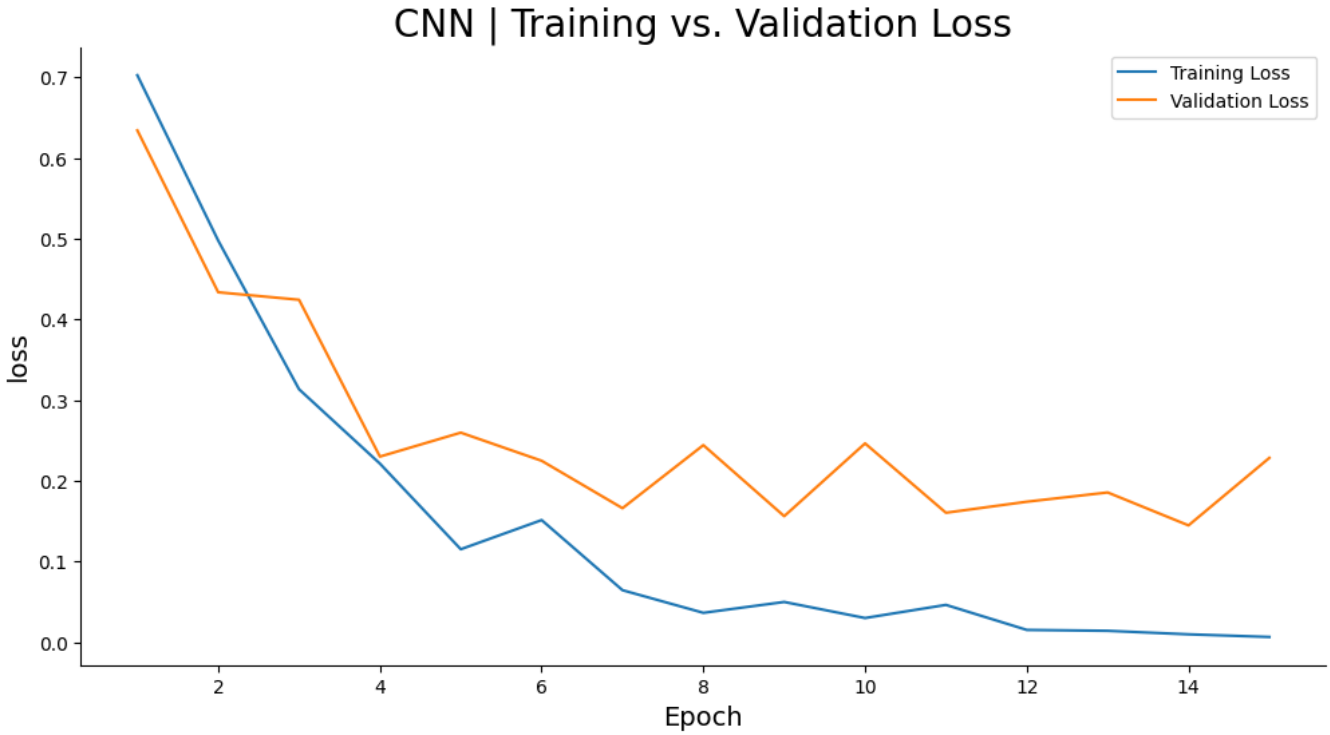
```
plt.plot(np.arange(1, 51), history_1.history['loss'], label='Training Loss')
plt.plot(np.arange(1, 51), history_1.history['val_loss'], label='Validation Loss
plt.title('MLP | Training vs. Validation Loss', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('loss', size=14)
plt.legend();
```



```
plt.plot(np.arange(1, 51), history_1.history['accuracy'], label='Training Accura
plt.plot(np.arange(1, 51), history_1.history['val_accuracy'], label='Validation
plt.title('MLP | Training vs. Validation Accuracy', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('Accuracy', size=14)
plt.legend();
```



```
plt.plot(np.arange(1, 16), history_2.history['loss'], label='Training Loss')
plt.plot(np.arange(1, 16), history_2.history['val_loss'], label='Validation Loss
plt.title('CNN | Training vs. Validation Loss', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('loss', size=14)
plt.legend();
```



```
plt.plot(np.arange(1, 16), history_2.history['accuracy'], label='Training Accura
plt.plot(np.arange(1, 16), history_2.history['val_accuracy'], label='Validation
plt.title('CNN | Training vs. Validation Accuracy', size=20)
plt.xlabel('Epoch', size=14)
plt.ylabel('Accuracy', size=14)
plt.legend();
```