Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Вычислительной математики и информационных технологий
Кафедра прикладной математики

Специальность: 01.03.04. Прикладная математика
Профиль: Прикладная математика

КУРСОВАЯ РАБОТА

Классификация рентген снимков легких с помощью нейронных сетей

Студент 4 курса
группы 09-822

«__»_____202_ г. _____ Юмагузин И.Д.

Научный руководитель
Старший преподаватель
«__»_____202_г. _____ Осипов Е.А.

Казань – 202_г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пневмония является одной из форм острой респираторной инфекции, при котором альвеолы, мелкие мешочки, из которых состоят легкие, заполняются гноем и жидкостью вместо воздуха, что делает дыхание болезненным и снижает поступление кислорода. Согласно Всемирной организации здравоохранения в 2017 году 808 694 детей до 5 лет умерли от пневмонии [1]. Основными методами диагностики пневмонии являются рентгенологическое исследование и исследование макроты, методм лечения – антибактериальная терапия.

Рисунок 1. Строение легких. Состояние альвеол у здорового человека слева и с пневмонией справа

Основная цель курсовой работы построить и обучить нейронную сеть классифицировать рентгент снимки здоровых легких и легкие с пневмонией. Примеры снимков приведены на рисунке 2. Задачи работы:

1. Изучить принципы построения полносвязной и сверточной

нейронных сетей

2. Реализовать и обучить полносвязную нейронную сеть классифицировать изображения.
3. Построить и обучить сверточные нейронные сети.
4. Сравнить предсказание обученных нейронных сетей на тестовой выборке.

Рисунок 2. Примеры классов из датасета [3]

ВВЕДЕНИЕ В НЕЙРОННЫЕ СЕТИ

Искусственные нейронные сети (ИНС) – это математическая модель, воплощающая работу биологических нейронных сетей живых организмов. То есть, ИНС – это попытка воссоздать работу мозга. Рассмотрим работу биологического нейрона. Хотя нейроны в головном мозге бывают разными, ее типичный представитель выглядит

следующим образом:

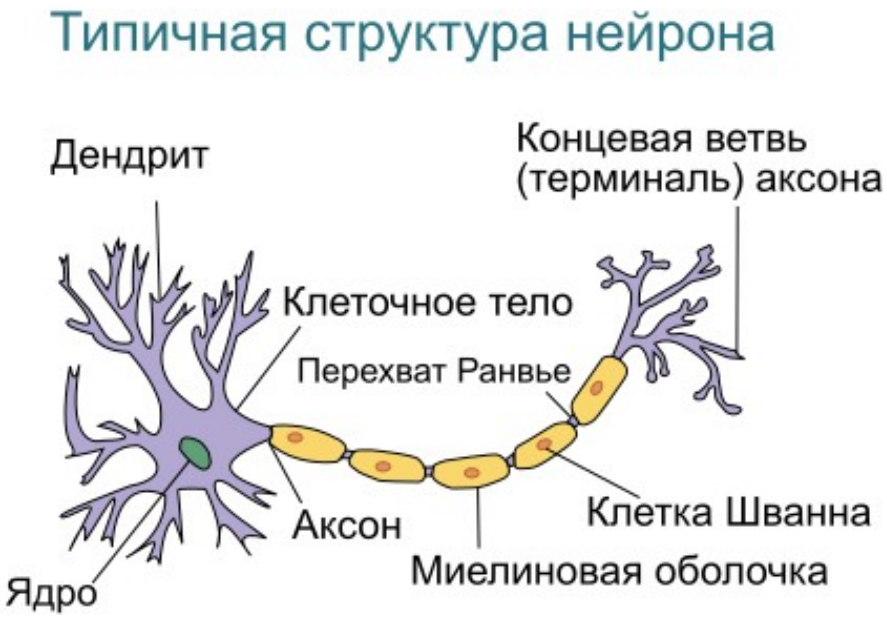

Рисунок 3. Строение биологического нейрона

У нейрона есть клеточное тело, дендриты, аксон и множество синапсов, с через которых проходят все сигналы к и от нейрона. В искусственном нейроне реализован схожий принцип приема и передачи сигнала. Математическая модель представлен на рисунке 4.

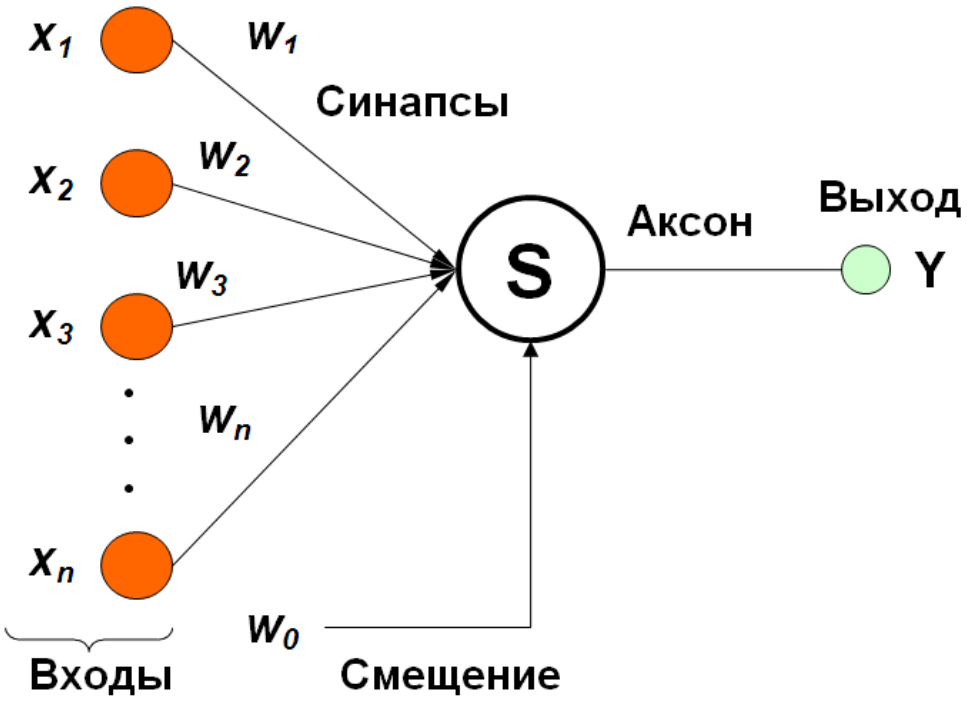

Рисунок 4. Искусственный нейрон

На вход подается вектор $X=(x_1, x_2, \ldots, x_n)$, далее вычисляется взвешенная сумма по весам и добавляется смещение:

$$S=\sum_{i=1}^{n} x_i w_i + w_0.$$

К сумме применяется функция активации:

$$Y=f(S).$$

Некторые применяемые на практике функции активации и их производные:

1. Логистическая (сигмоида):

$$f(x)=\sigma(x)=\frac{1}{1+e^{-x}}, \quad f'(x)=f(x)(1-f(x)), \quad \text{область значений}:(0;1)$$

2. ReLU (Rectified linear unit):

$$f(x)=\begin{cases}0, & x<0\\ x, & x\geq 0\end{cases},\quad f'(x)=\begin{cases}0, & x<0\\ 1, & x\geq 0\end{cases},\quad \text{область значений}:[0,\infty)$$

3. Leaky ReLU (Leaky rectified linear unit):

$$f(x)=\begin{cases}0.01x, & x<0\\ x, & x\geq 0\end{cases},\quad f'(x)=\begin{cases}0.01, & x<0\\ 1, & x\geq 0\end{cases},\quad \text{область значений}:(-\infty,\infty).$$

Инициализация весов *W* обычно присходит из нормального распределения со нулевым средним и дисперсией $\frac{2}{n}$, где *n* – размер входного вектора.

ПОЛНОСВЯЗНЫЕ НЕЙРОННЫЕ СЕТИ

Полносвязная нейронная сеть (fully connected neural network)– это нейронная сеть, в которой каждый нейрон *l*-го слоя связан с каждым нейроном (*l-1*)-го слоя. На рисунке 5 представлена архитектура полносвязной нейронной сети.

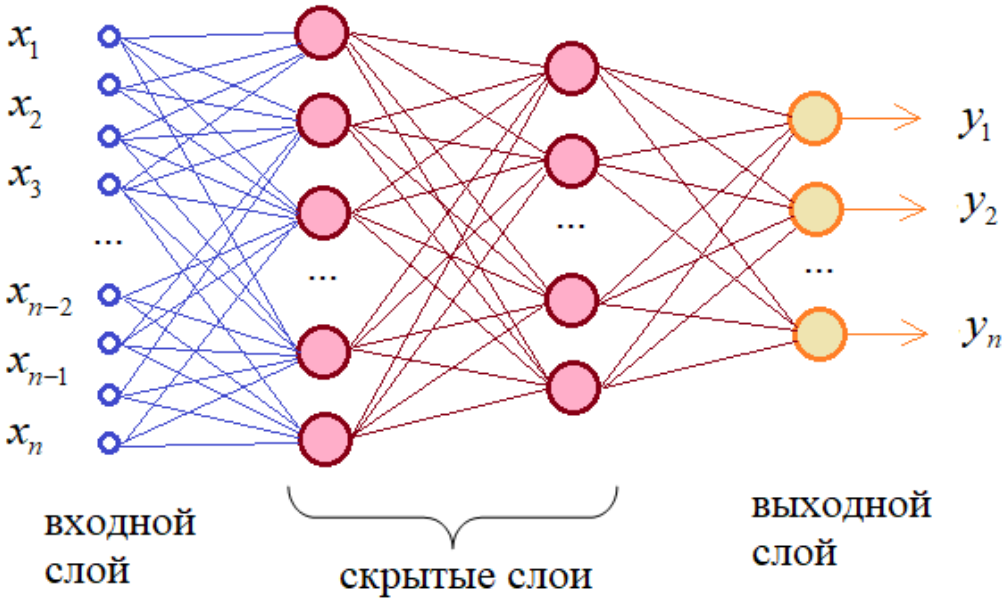

Рисунок 5. Структура полносвязной нейронной сети.

Для задачи классификации количество нейронов в последнем слое будет равна количеству классов в датасете.

Обучение нейронной сети происходит в два этапа:

- Прямое распространение

  Для каждого нейрона на l-ом слое вычисляется активация:

  $$a_i^{(l)} = \bar{w}_i^{(l)T} \bar{z}^{(l-1)} + b_i^{(l)}$$

  Далее вычисляется выход по нейронам:

  $$\bar{z}^{(l)} = \bar{f}\left(\bar{a}^{(l)}\right)$$

  После прохождения по всем слоям вычисляются значения функции Softmax:

  $$y(\bar{z}) = Softmax(\bar{z}) = \left(\frac{e^{z_1}}{\sum_i e^{z_i}}, \frac{e^{z_2}}{\sum_i e^{z_i}}, \cdots, \frac{e^{z_n}}{\sum_i e^{z_i}} \right).$$

Значения *y* являются предсказаниями для каждого класса. Можно заметить, что

$$\sum_{i=1}^{n} y_i = \frac{e^{z_1}}{\sum_i e^{z_i}} + \frac{e^{z_2}}{\sum_i e^{z_i}} + \cdots \frac{e^{z_n}}{\sum_i e^{z_i}} = \frac{\sum_i e^{z_i}}{\sum_i e^{z_i}} = 1.$$

Вычисляется функция потерь:

$$E(\bar{w}) = -\sum_{i=1}^{N} \sum_{k}^{K} t_i^{(k)} log\, y_k(x_i, w),$$

где K – количество классов, x_i – элемент из обучающей выборки и $t_i^{(l)}$ - это one-hot encoding вектор:

$$t_i^{(k)} = \begin{cases} 1, & k = t_i \\ 0, & k \neq t_i \end{cases},$$

где k – это настоящий класс, к которому принадлежит элемент x_i.

Тогда для каждого отдельного элемента:

$$E = -\sum_{k}^{K} t^{(k)} log\, y_k(\bar{x}).$$

- Обратное распространение

  Обратное распространение (Back propogation) основано на цепном правиле дифференцирования:

  $$(f(g(x)))'_x = (f(g(x)))'_g \cdot g(x)'_x \quad .$$

  Например, нам известны градиент функции потерь по *y*, функция *y(x)*, необходимо найти градиет функции потерь по *x*:

итак,

$$\nabla_{\bar{y}} E = \left(\frac{\partial E}{\partial y_1}, \frac{\partial E}{\partial y_2}, \cdots, \frac{\partial E}{\partial y_n} \right),$$

тогда

$$\nabla_{\bar{x}} E = \left(\sum_i \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_1}, \sum_i \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_2}, \cdots, \sum_i \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_n} \right),$$

то есть,

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_n} \end{pmatrix} = \left(\frac{\partial \bar{y}}{\partial x_1}, \frac{\partial \bar{y}}{\partial x_2}, \cdots, \frac{\partial \bar{y}}{\partial x_n} \right).$$

Отсюда получаем, что

$$\nabla_{\bar{x}} E = \left(\frac{\partial y}{\partial x} \right)^T \cdot \nabla_{\bar{y}} E.$$

По такому принципу находим градиенты в обратном направлении.

То есть,

$$\begin{array}{l} \nabla_{\bar{y}} E \to \nabla_{\bar{z}^{(L)}} E \to \nabla_{\bar{a}^{(L)}} E \to \nabla_{\bar{w}^{(L)}} E \to \cdots \to \nabla_{\bar{w}^{(1)}} E \\ \qquad\qquad\qquad\qquad \searrow \\ \qquad\qquad\qquad\qquad\quad \nabla_{\bar{b}^{(L)}} E \end{array}.$$

Формулы для вычисления градиентов:

$$\nabla_{\bar{z}^{(L)}} E = \bar{y} - \bar{t}\,,$$
$$\nabla_{\bar{a}^{(l)}} E = f'\left(a^{(l)}\right) \circ \nabla_{\bar{z}^{(L)}} E\,,$$
$$\nabla_{w^{(l)}} E = \nabla_{\bar{a}^{(l)}} E \cdot \left(\bar{z}^{(l-1)}\right)^T,$$

и наконец,

$$\nabla_{\bar{z}^{(l-1)}} E = \left(W^{(l)}\right)^T \cdot \nabla_{\bar{a}^{(l)}} E\,.$$

СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Свёрточная нейронная сеть (Convolutional Neural Network, CNN) – архитектура искусственных глубоких нейронных сетей, предложанная Яном Лекуном в 1988 году. В данной архитектуре реализованы некоторые особенности зрительной коры, что позволяет CNN эффективно расспознавать образы [2]. В основе лежит операция свёртки фрагмента изображения с ядром (фильтром), то есть, фрагмент изображения поэлементно умножается на ядро, результат суммируется и присваивается в соответствующую позицию. На рисунке 6 представлена операция свертки с параметром *stride* равной 1, это означает, что фильтр будет перемещаться по входному изображению с шагом в один пиксель.

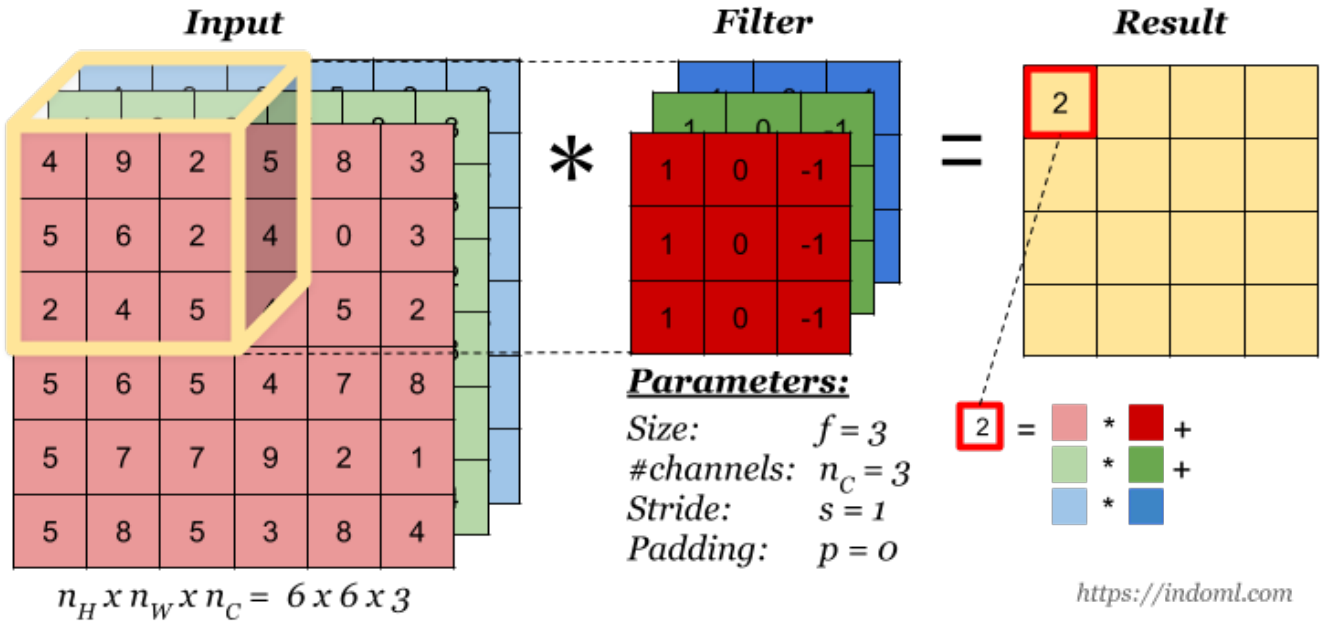

Рисунок 6. Операция свёртки RGB изображения с фильтром с 3 каналами. Результат: матрица глубиной 1

Карта активации (выходного изображения) будет иметь размерность

$$\lfloor \frac{w-f}{s}+1 \rfloor \times \lfloor \frac{h-f}{s}+1 \rfloor,$$

где w – ширина входного изображения, h – ее высота, f – размер фильтра, s – коэффициент сдвига. Другим важным методом является операция MaxPooling:

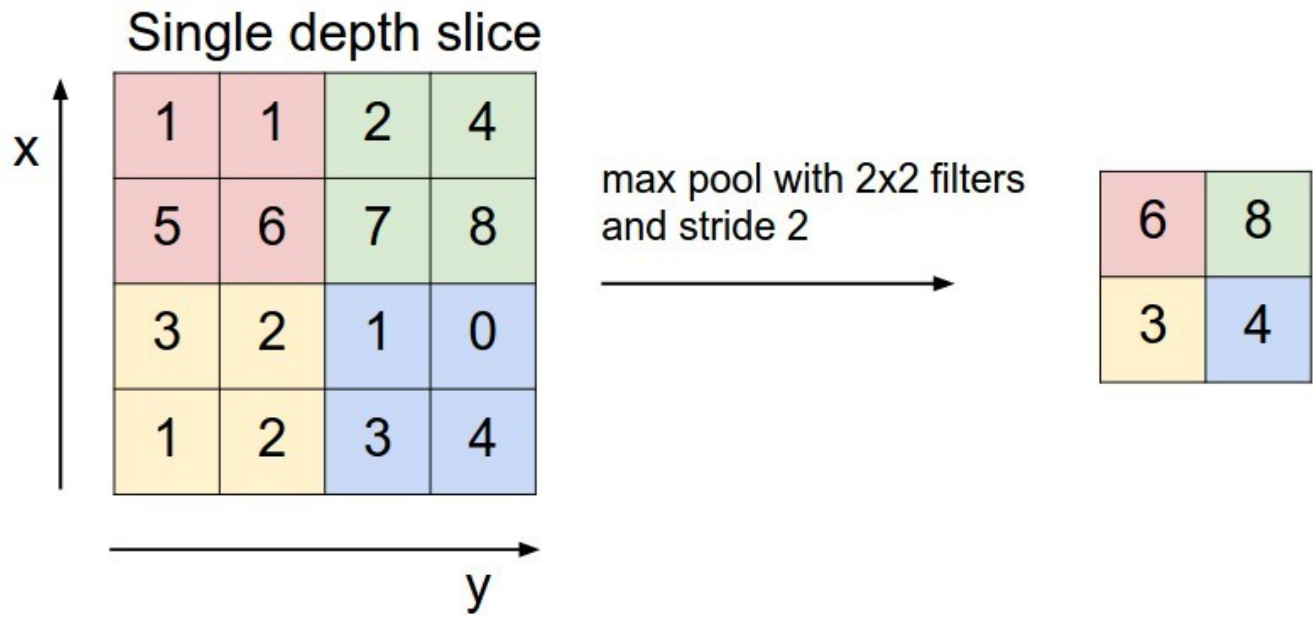

Рисунок 7. Операция MaxPooling, для отдельного канала

После свёрточных слоев, выходной тензор на последнем свёрточном слое передается входными данными в полносвязную нейронную сеть.

- Прямое распространение

$$a^{(l,r)} = Z^{(l-1)} * W^{(l,r)} + b_r^{(l)} \cdot J, \text{ где } J = \{1\}_{m_l \times n_l},$$
$$Z^{(l)} = f\left(a^{(l)}\right),$$

  где “*” означает операцию свертки.

- Обратное распространение

  Формулы для вычисления градиентов:

$$\nabla_{a^{(l)}} E = f'\left(a^{(l)}\right) \circ \nabla_{\bar{z}^{(L)}} E,$$
$$\nabla_{w^{(l,r)}} E = Z^{(l-1)} * \nabla_{\bar{a}^{(l,r)}} E,$$
$$\nabla_{b_r^{(l)}} E = \sum \left(\nabla_{a^{(l,r)}} E\right), r = \overline{1, r_l}$$
$$\nabla_{Z^{(l-1,k)}} E = ZeroPad_{\tilde{m}_l - 1}\left(\nabla_{\bar{a}^{(l)}} E\right) * U^{(l,r)},$$

где $\widetilde{m}_l - 1$ размер фильтра, а $U_r^{(l,k)} = \widetilde{W}_k^{(l,r)}$, то есть, *k*-ый канал *r*-го фильтра, тогда $U^{(l,k)}$ - это *k*-ые каналы всех фильтров *l*-го слоя. $\widetilde{W}_k^{(l,r)}$

- это перевернутая по столбцам, затем по строкам фильтр.

Архитектура CNN может быть построена следующим образом:

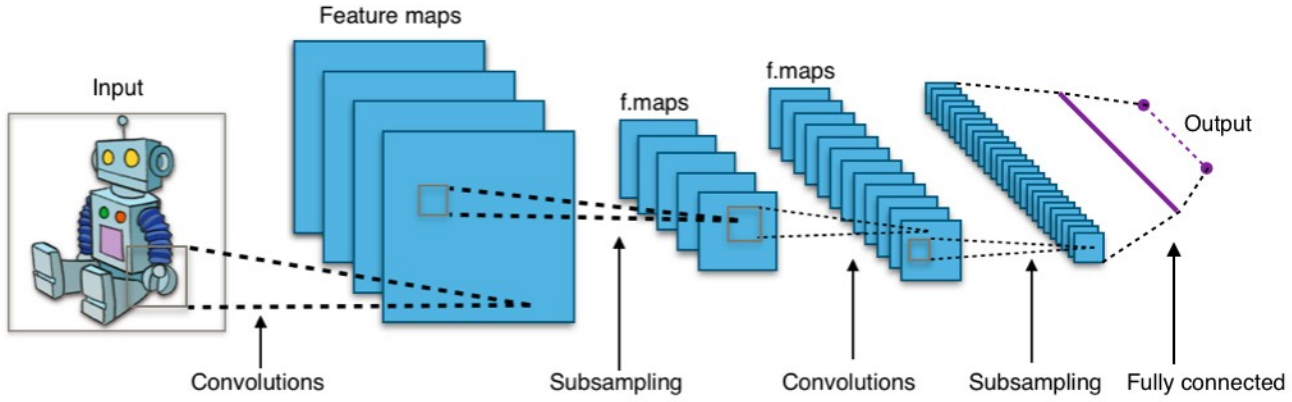

Рисунок 8. Архитектура свёрточной нейронной сети

СРАВНЕНИЕ РЕЗУЛЬТАТОВ

В рамках курсового проекта реализовал полносвязную нейронную сеть, которая принимает на вход вектор размером в 150*150, для этого необходима уменьшить разрешение входного изображние до 150 пикселей в ширину и столько же в длину.

В последнем слое два нейрона, так как всего два класса. В итоге нейронная сеть имеет следующую архитектуру:

Layer (type)	Output Shape
Fully-Connected	(22500, 100)
Fully-Connected	(100, 50)
Fully-Connected	(50, 2)

Результат обучения на тренеровычных данных за 3 эпохи:

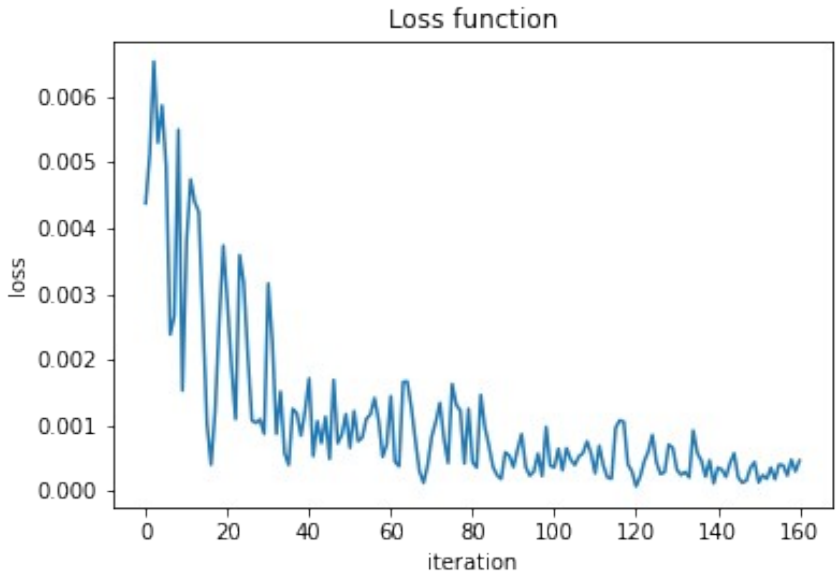

Рисунок 9. Обучение полносвязной нейронной сети на 3 эпохи

При этом точность, который определяется как отношение количества правильных предсказаний и общее количество предсказаний:

Accuracy: 0.8028846153846154

Свёрточная нейронная сеть:

Model: "sequential"

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 150, 150, 3)       0
```

```
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 32)       0
 2D)

 flatten (Flatten)           (None, 9248)              0

 dense (Dense)               (None, 128)               1183872

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,203,522
Trainable params: 1,203,522
Non-trainable params: 0
_________________________________________________________________
```

Результат за пять эпох обучения на тренеровочных данных:

loss: 0.7941 - accuracy: 0.8173

ЗАКЛЮЧЕНИЕ

Использование технологий искусственного интелекта в медецине может помочь в постановке диагнозов, предупреждении развития заболеваний, особенно связанные с новообразованиями в различных органах, в том числе и в легких.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] https://ru.wikipedia.org/wiki/Пневмония

[2] https://en.wikipedia.org/wiki/Convolutional_neural_network

[3] https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

https://www.youtube.com/playlist?list=PL6-BrcpR2C5QrLMaIOstSxZp4RfhveDSP

ЛИСТИНГ КОДА

```
import numpy as np
import os
from cv2 import cv2
import h5py
import matplotlib.pyplot as plt
def relu(a):
    z = np.zeros_like(a)
    return np.maximum(a, z)

def diff_relu(a):
    z = np.zeros_like(a)
    z[a > 0] = 1
    return z

def softmax(Z):
    Z = np.array(Z, dtype=np.float128)
    return np.exp(Z) / np.sum(np.exp(Z))

def check_point(model, ep, error):
    f = h5py.File('mlp_model_3_for_xray_ds_learned.hdf5', 'a')
    grp = f.create_group(f"epoch_{ep}")
    for i in range(len(model)):
        grp_l = grp.create_group(f"layer_{i}")
        dset = grp_l.create_dataset("weight", data=model[i][0])
        dset = grp_l.create_dataset("bias", data=model[i][1])
    dse = grp.create_dataset('error', data=error)
```

```
    f.close
def check_point_return(model, ep):
    f = h5py.File('mlp_model_for_xray_ds_learned.hdf5', 'r')
    epoch_grp = f[f"epoch_{ep}"]
    for i in range(len(model)):
        layer_grp = epoch_grp[f"layer_{i}"]
        model[i][0] = layer_grp['weight']
        model[i][1] = layer_grp['bias']
    error = epoch_grp['error']
    f.close
    return model, error
labels = ['PNEUMONIA', 'NORMAL']
img_size = 150

def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) # Reshaping images to
preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
```

```
    np.random.shuffle(data)
    return np.array(data)
train_data = get_training_data('/home/ilnar/Documents/Jupyter-notebooks/datasets/
chest_xray/chest_xray/train')
test_data = get_training_data('/home/ilnar/Documents/Jupyter-notebooks/datasets/
chest_xray/chest_xray/test')
val_data = get_training_data('/home/ilnar/Documents/Jupyter-notebooks/datasets/
chest_xray/chest_xray/val')
X_train, y_train = train_data[:, 0], train_data[:, 1]
X_test, y_test = test_data[:, 0], test_data[:, 1]
for i in range(len(X_train)):
    X_train[i] = X_train[i].reshape(150*150,)

for i in range(len(X_test)):
    X_test[i] = X_test[i].reshape(150*150,)
X_train, X_test = X_train / 255.0, X_test / 255.0
def layer(input_layer_count, output_layer_count):
    W = np.random.normal(0, 2/100, (output_layer_count, input_layer_count))
    b = np.ones((output_layer_count))
    print(W.shape)
    return [W, b]
def feed_forward(model, z):
    zs = [z]
    activations = []
    for i in range(len(model)):
        W, b = model[i]
        a = np.dot(W, zs[i]) + b
```

```
        z = relu(a)

        zs.append(z)
        activations.append(a)
    return activations, zs

def backpropagation(model, activations, zs, X_, grad_zlE, lr):
    for i in range(len(model) - 1, -1, -1):
        if i == 0:
            [W, b], a, z = model[i].copy(), activations[i], zs[i + 1]
            grad_aE = diff_relu(a) * grad_zlE
            grad_bE = grad_aE
            b = b - lr * grad_bE
            grad_w0E = grad_aE[np.newaxis].T * X_[np.newaxis]
            W = W - lr * grad_w0E
            model[i] = [W, b]
            break
        [W, b], a, z = model[i].copy(), activations[i], zs[i + 1]
        z_l_1 = zs[i]

        grad_aE = diff_relu(a) * grad_zlE

        grad_bE = grad_aE
        b = b - lr * grad_bE

        grad_wE = np.dot(grad_aE[np.newaxis].T, z_l_1[np.newaxis])
```

```
        W = W - lr * grad_wE

        grad_z_l_1E = np.dot(W.T, grad_aE)
        grad_zlE = grad_z_l_1E

        model[i] = [W, b]
    return model

def train(model, X_train, y_train, learning_rate=0.00001, epoch=1, record=False):
    step = 20
    lr = learning_rate
    E_xs = np.array([])
    try:
        for ep in range(epoch):
            print("epoch:", ep)
            index = 0
            E_xi = 0
            for l in range(len(X_train)):
                X_ = X_train[l]
                y_ = y_train[l]
                z = X_

                activations, zs = feed_forward(model, z)

                y = softmax(zs[-1])

                t = [1 if i == y_ else 0 for i in [0, 1]]
```

```
            E_xi = (- np.dot(t, np.log(y)))
            E_xs = np.append(E_xs, E_xi)

            #if y_ == 1:
            #    print(f"{index} iteration:", y_, ":", y)
            if index % step == 0:
                print(f"{index} iteration: y = {y_}, predict {y}, loss: {E_xi}")
                #print(f"{index} iteration: y={y_}, predict {y}, Max loss in ({index - step}, {index}): {E_xs[-step:].max()}")

            i = t.index(1)

            grad_yE = np.zeros_like(y)
            grad_yE[i] = -1 / y[i]

            grad_zlE = (y - t)

            model = backpropagation(model, activations, zs, X_, grad_zlE, lr)

            index += 1
            #if index == early_stop:
            #    return model, E_xs
        check_point(model, ep, E_xs)
    except KeyboardInterrupt:
        if record:
            check_point(model, f'eary_stopped', E_xs)
        print("Training is early stopped!")
```

```
            return model, E_xs
        return model, E_xs
model = [
    layer(img_size**2, 100),
    layer(100, 50),
    layer(50, 2)
]
model_1 = model.copy()
model_1, E_xs = train(model_1, X_train, y_train, learning_rate=0.0001, epoch=5)
def np_max(l):
    max_idx = np.argmax(l)
    max_val = l[max_idx]
    return (max_idx, max_val)

def metrics(model, X_test, y_test):
    correct = 0
    total = len(y_test)
    for l in range(len(X_test)):
        X_ = X_test[l]
        y_ = y_test[l]
        t = [1 if i == y_ else 0 for i in [0, 1]]
        i = t.index(1)
        z = X_
        for layer in model:
            W, b = layer.copy()
            a = np.dot(W, z) + b
            z = relu(a)
```

```
        y = softmax(z)

        ind, val = np_max(y)
#         print(y_, ":", y[i], f"predict for {labels[ind]}:",  val)
        if ind == i:
            correct += 1
    print(f"Accuracy: {correct/total}")

metrics(model_1, X_test, y_test)
loss = np.array([])
step = 100
for i in range(step, len(E_xs), step):
    loss = np.append(loss, E_xs[i-step:i].min())
print(loss.size)
plt.plot(np.arange(len(loss)), loss)
plt.title("Loss function")
plt.ylabel("loss")
plt.xlabel("iteration")
plt.savefig("mlp_lear_loss")
```

Свёрточная нейронная сеть:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import os
from cv2 import cv2
import matplotlib.pyplot as plt
```

```
import matplotlib.image as image
image_size = 150
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "datasets/chest_xray/train/",
    seed=42,
    image_size=(image_size, image_size),
    batch_size=32,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "datasets/chest_xray/val/",
    seed=42,
    image_size=(image_size, image_size),
    batch_size=32,
)
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "datasets/chest_xray/test/",
    seed=42,
    image_size=(image_size, image_size),
    batch_size=32,
)
for image_batch, label in train_ds.take(1):
    image = image_batch[23].numpy().astype('uint8')
    image_label = label[23].numpy()

    plt.title(labels[image_label])
    plt.imshow(image)
```

```
    image = image_batch[23].numpy().astype('uint8')
    image_label = label[23].numpy()

    plt.title(labels[image_label])
    plt.imshow(image)

def plotImages(images_arr, label):
    fig, axes = plt.subplots(1, 3, figsize=(15, 15))
    for img, ax, l in zip( images_arr, axes, label):
        ax.imshow(img)
        ax.set_title(labels[l])
        ax.axis('off')
    plt.savefig("обарзцы классов")
    plt.show()

for image_batch, labels_batch in train_ds:
    plotImages(image_batch[10:13].numpy().astype('uint8'),
labels_batch[10:13].numpy())
    print(image_batch.shape)
    print(labels_batch.shape)
    break
model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
```

```
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(2)
])
model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
model.summary()
model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=5
)
```