# Министерство науки и высшего образования Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»



## Кафедра теоретической и прикладной информатики

### Лабораторная работа №5 по дисциплине «Основы теории машинного обучения»

**КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студенты: | Ершов П.К.  Малышкина Е.Д.  Слободчикова А.Э. | | |
| Вариант: | 4 |  |  |
| Преподаватель: | Попов А.А. |  |  |
|  |  |  |  |  |

Новосибирск

2021

1. **Цель.**

Получить практические навыки по решению задачи классификации изображений с применением свёрточных нейронных сетей (CNN).

1. **Содержание работы.**
2. Ознакомление с теоретическими основами используемых архитектур нейронных сетей (НС).
3. Ознакомление с возможностями их реализации в рамках свободных библиотек типа TensorFlow.
4. Поиск или формирование необходимых датасетов для выбранной задачи.
5. Кодирование и отладка программы. Обучение и тестирование НС.
6. Написание отчета.
7. Защита лабораторной работы.
8. **Теоретическая часть**

Свёрточные нейронные сети (CNN) – это специальная архитектура нейронных сетей, предложенная Яном Лекуном в 1988 для распознавания образов. Суть архитектуры в чередовании свёрточных слоёв, то есть слоёв, которые выделяют определённые признаки, и слоёв пулинга, то есть слоёв, которые уплотняют данные до менее подробных и тем самым позволяют не переобучаться сети.

В данной работе используется сеть из 13 слоёв.

В качестве слоёв свёртки используются 2 последовательно расположенных Conv2D.

В качестве слоёв пулинга используются MaxPool2D, идущий сразу за слоями свёртки.

В качестве слоя активации используется Dropout, которые помогает решить проблему переобучения.

Функцией активации свёрточных слоёв выберем ReLu, так как она является хорошим аппроксиматором.

Для выходного слоя функцией активации выберем softmax, так как её выходные значения нейронов зависят от суммы предыдущих нейронов.

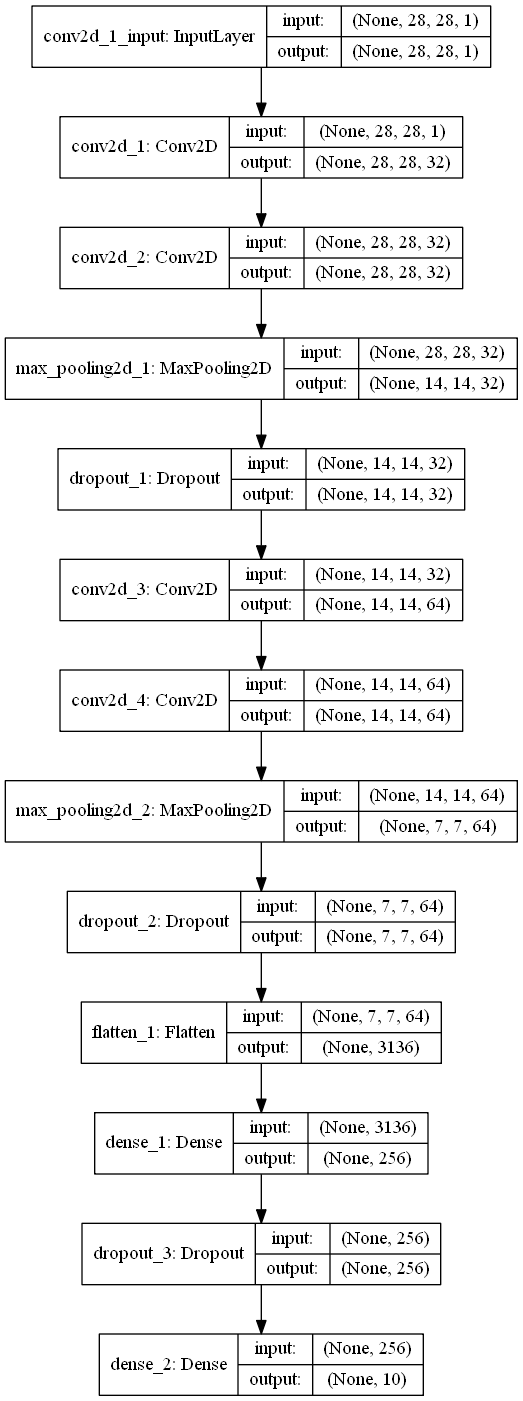


Рисунок 1. Структура нейронной сети

Обучение будет проводиться с помощью MNIST - набора данных, представляющих рукописные цифровые изображения.

Число эпох 10. Число батчей в эпохах 256.

1. **Ход работы**

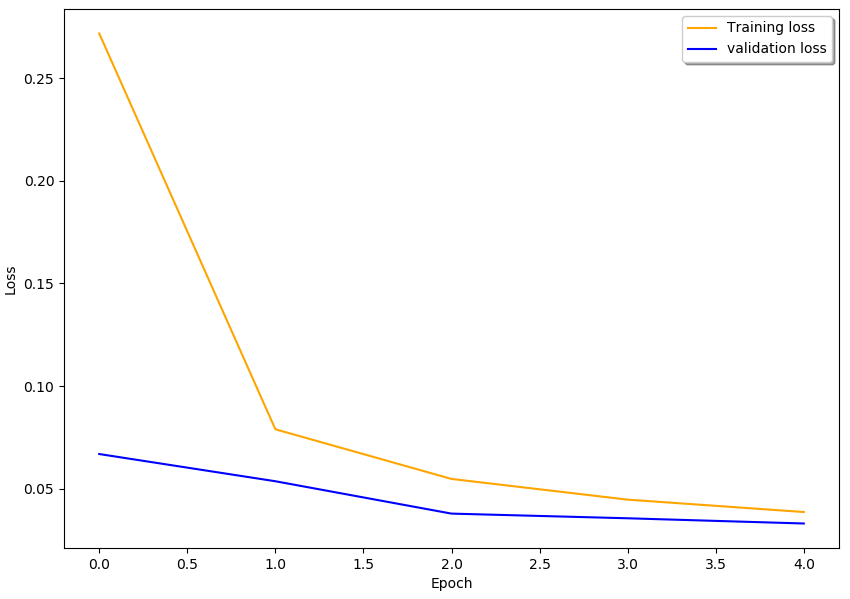


Рисунок 2. График функции потерь (синим – тренировочная выборка, оранжевым – верификационная)

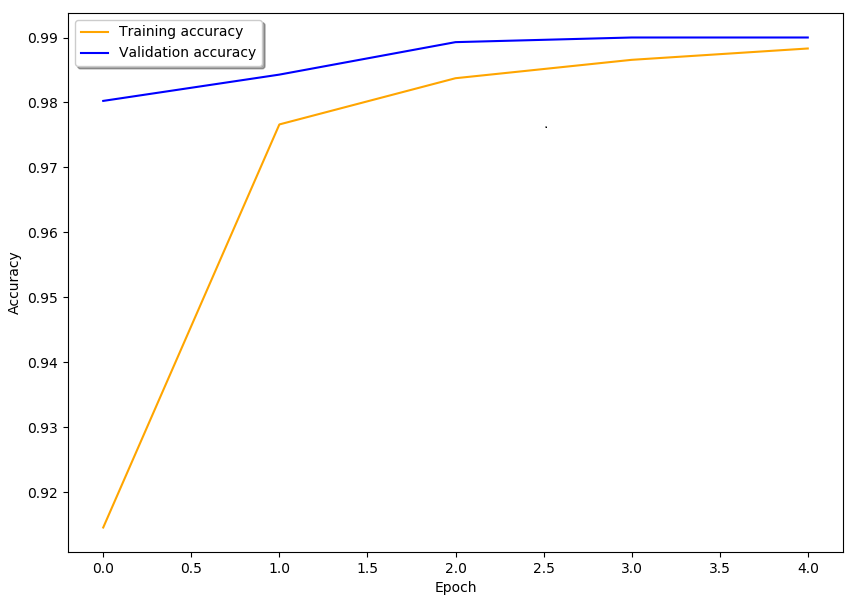


Рисунок 3. График точности

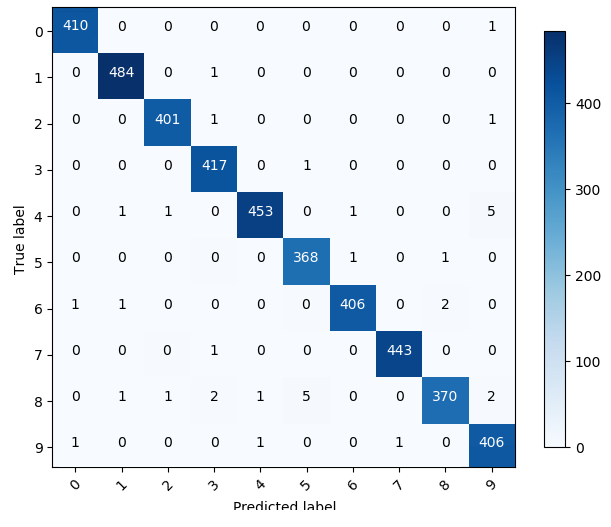


Рисунок 4. Матрица ошибок.

Исходя из результатов в матрице ошибок можно сделать вывод, что модель достаточно точная. Наблюдаются некоторые затруднения с 8 и 9, что подтверждается данными по ошибками на рисунке 5.

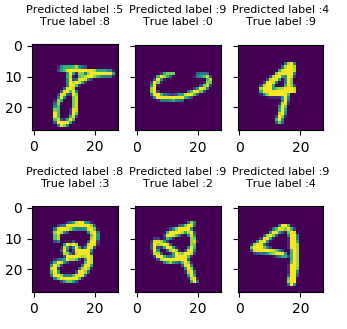


Рисунок 5. Ошибки классификации

1. **Текст программы:**

**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.image **as** mpimg  
**import** seaborn **as** sns  
  
np.random.seed(2)  
  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** confusion\_matrix  
**import** itertools  
  
**from** keras.utils.np\_utils **import** to\_categorical *# convert to one-hot-encoding***from** keras.models **import** Sequential  
**from** keras.layers **import** Dense, Dropout, Flatten, Conv2D, MaxPool2D  
**from** keras.optimizers **import** RMSprop  
**from** keras.preprocessing.image **import** ImageDataGenerator  
**from** keras.callbacks **import** ReduceLROnPlateau  
**from** keras.utils.vis\_utils **import** plot\_model *# Загрузка данных*train = pd.read\_csv(**"train.csv"**)  
test = pd.read\_csv(**"test.csv"**)  
Y\_train = train[**"label"**]  
  
*# Отбросить столбец с надписью "label"*X\_train = train.drop(labels = [**"label"**],axis = 1)  
  
**del** train  
  
Y\_train.value\_counts()  
  
*# Проверка данных*X\_train.isnull().any().describe()  
test.isnull().any().describe()  
  
*# Нормализуем данные*X\_train = X\_train / 255.0  
test = test / 255.0  
  
*# Изменение формы изображения в 3-х измерениях (высота = 28 пикселей, ширина = 28 пикселей, канал = 1)*X\_train = X\_train.values.reshape(-1,28,28,1)  
test = test.values.reshape(-1,28,28,1)  
  
*# Кодировать метки в один вектор*Y\_train = to\_categorical(Y\_train, num\_classes = 10)  
  
random\_seed = 2  
  
*# Разделение тренировочного набора (90%) и набора проверки (10%)*X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X\_train, Y\_train, test\_size = 0.1, random\_state=random\_seed)  
  
*# Пример изучаемого объекта*g = plt.imshow(X\_train[0][:,:,0])  
  
*# Установка слоев модели*model = Sequential()  
  
model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = **'Same'**, activation =**'relu'**, input\_shape = (28,28,1)))  
model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = **'Same'**, activation =**'relu'**))  
model.add(MaxPool2D(pool\_size=(2,2)))  
model.add(Dropout(0.25))  
  
  
model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = **'Same'**, activation =**'relu'**))  
model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = **'Same'**, activation =**'relu'**))  
model.add(MaxPool2D(pool\_size=(2,2), strides=(2,2)))  
model.add(Dropout(0.25))  
  
  
model.add(Flatten())  
model.add(Dense(256, activation = **"relu"**))  
model.add(Dropout(0.5))  
model.add(Dense(10, activation = **"softmax"**))  
  
*# Опредление оптимизации*optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)  
  
model.compile(optimizer = optimizer , loss = **"categorical\_crossentropy"**, metrics=[**"accuracy"**])  
plot\_model(model, to\_file=**'model\_plot.png'**, show\_shapes=**True**, show\_layer\_names=**True**)  
  
learning\_rate\_reduction = ReduceLROnPlateau(monitor=**'val\_accuracy'**, patience=3, verbose=1, factor=0.5, min\_lr=0.00001)  
epochs = 10  
batch\_size = 256  
  
history = model.fit(X\_train, Y\_train, batch\_size = batch\_size, epochs = epochs,  
 validation\_data = (X\_val, Y\_val), verbose = 2)  
  
  
**def** plot\_loss(history):  
  
 plt.figure(figsize=(10, 7))  
  
 plt.plot(history.history[**'loss'**], color=**'orange'**, label=**"Training loss"**)  
 plt.plot(history.history[**'val\_loss'**], color=**'b'**, label=**"validation loss"**)  
 plt.legend(loc=**'best'**, shadow=**True**)  
 plt.xlabel(**'Epoch'**)  
 plt.ylabel(**'Loss'**)  
 plt.savefig(**'loss\_plot.png'**)  
  
  
**def** plot\_acc(history):  
  
 plt.figure(figsize=(10, 7))  
 plt.plot(history.history[**'accuracy'**], color=**'orange'**, label=**"Training accuracy"**)  
 plt.plot(history.history[**'val\_accuracy'**], color=**'b'**,label=**"Validation accuracy"**)  
 plt.legend(loc=**'best'**, shadow=**True**)  
 plt.xlabel(**'Epoch'**)  
 plt.ylabel(**'Accuracy'**)  
 plt.savefig(**'accuracy\_plot.png'**)  
  
  
**def** plot\_confusion\_matrix(cm, classes, normalize=**False**, title=**'Confusion matrix'**, cmap=plt.cm.Blues):  
 plt.imshow(cm, interpolation=**'nearest'**, cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 **if** normalize:  
 cm = cm.astype(**'float'**) / cm.sum(axis=1)[:, np.newaxis]  
  
 thresh = cm.max() / 2.  
 **for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, cm[i, j],  
 horizontalalignment=**"center"**,  
 color=**"white" if** cm[i, j] > thresh **else "black"**)  
  
 plt.tight\_layout()  
 plt.ylabel(**'True label'**)  
 plt.xlabel(**'Predicted label'**)  
 plt.savefig(**'loss\_matrix.png'**)  
  
*# Предсказание значения из набора данных проверки*Y\_pred = model.predict(X\_val)  
*# Преобразование классов прогнозов в одни векторы*Y\_pred\_classes = np.argmax(Y\_pred,axis = 1)  
*# Преобразование проверочных наблюдений в одни векторы*Y\_true = np.argmax(Y\_val,axis = 1)  
*# Вычисление матрицы ошибок*confusion\_mtx = confusion\_matrix(Y\_true, Y\_pred\_classes)  
*# Построение матрицы ошибок*plot\_confusion\_matrix(confusion\_mtx, classes = range(10))  
  
errors = (Y\_pred\_classes - Y\_true != 0)  
  
Y\_pred\_classes\_errors = Y\_pred\_classes[errors]  
Y\_pred\_errors = Y\_pred[errors]  
Y\_true\_errors = Y\_true[errors]  
X\_val\_errors = X\_val[errors]  
  
**def** display\_errors(errors\_index,img\_errors,pred\_errors, obs\_errors):  
 n = 0  
 nrows = 2  
 ncols = 5  
 fig, ax = plt.subplots(nrows,ncols,sharex=**True**,sharey=**True**)  
 **for** row **in** range(nrows):  
 **for** col **in** range(ncols):  
 error = errors\_index[n]  
 ax[row,col].imshow((img\_errors[error]).reshape((28,28)))  
 ax[row,col].set\_title(**"Predicted label :{}\nTrue label :{}\n"**.format(pred\_errors[error],obs\_errors[error]), fontdict={**'fontsize'**: 8, **'fontweight'**: **'medium'**})  
 n += 1  
 plt.savefig(**'error.png'**)  
  
*# Вероятности неверных предсказанных чисел*Y\_pred\_errors\_prob = np.max(Y\_pred\_errors,axis = 1)  
  
*# Прогнозируемые вероятности истинных значений в наборе ошибок*true\_prob\_errors = np.diagonal(np.take(Y\_pred\_errors, Y\_true\_errors, axis=1))  
  
*# Разница между вероятностью предсказанного значения и истинным*delta\_pred\_true\_errors = Y\_pred\_errors\_prob - true\_prob\_errors  
  
*# Отсортированный список ошибок*sorted\_dela\_errors = np.argsort(delta\_pred\_true\_errors)  
  
*# 6 ошибок*most\_important\_errors = sorted\_dela\_errors[-10:]  
  
*# Демонстрация данных ошибок*display\_errors(most\_important\_errors, X\_val\_errors, Y\_pred\_classes\_errors, Y\_true\_errors)  
  
*# Помещение результатов предсказаний в файл .csv  
# Предсказанные результаты*results = model.predict(test)  
  
results = np.argmax(results,axis = 1)  
  
results = pd.Series(results,name=**"Label"**)  
  
submission = pd.concat([pd.Series(range(1,28001),name = **"ImageId"**),results],axis = 1)  
  
submission.to\_csv(**"cnn\_mnist\_datagen.csv"**,index=**False**)  
  
plot\_loss(history)  
plot\_acc(history)