Deep Learning Examination

1st 999995
dept. Oslo, Norway
Artificial Intelligence
Oslo, Norway

2nd 999301
dept. Oslo, Norway
Artificial Intelligence
Oslo, Norway

line 1: 3rd 112015
dept. Oslo, Norway
Artificial Intelligence
Oslo, Norway

***Abstract* – Deep learning, a subset of machine learning, enables computers to learn from examples. This approach facilitates the execution of classification tasks on various data types, including images, text, and sound, with notable accuracy. The effectiveness of deep learning models is largely attributed to training on extensive datasets.**

I. Introduction

This paper addresses specific examination tasks within the PGR207 (Deep Learning) course. The focus is on three distinct aspects: different models, activation functions, and optimization methods. The Kvasir-Instrument dataset, comprising 590 annotated frames of gastrointestinal procedure tools, serves as the basis for training and testing these models. Key methodologies include the use of a learning-rate scheduler and the early stopping method, with a patience parameter set to three, to optimize model checkpoints. Additionally, image data is preprocessed for uniformity and converted into arrays, enhancing model compatibility.

II. Three Different Optimization Methods (Results)

A. U-Net

The model

The first part of the unet model encodes the image. It consists of 3x3 convolutional layers using Relu activation and he-normal initialization followd by a 2x2 max pooling operation. The next technique used is the dropout, which is used to avoid overfitting by limiting the ability to "memorize" the "answers", which is set to 0.2 for the first two blocks and progresses to 0.3 for the third one.

The second part of the unet model decodes the image. Similar to the encoding, the decoding uses 3x3 convolutional layers with relu and he-normal. However instead of 2d, it uses transposed convolutional layers for the feature maps. For the output it uses sigmoid activation.

Training

It is compiled with Adam and an initial learning rate of 0.001, and uses binary cross-entropy as the loss. To prevent overfitting, early stopping is incorporated. The model stops after the 10th epoch where the validation loss has not improved. To best show the accuracy, I have IoU, F1 score and pixel accuracy.

Results

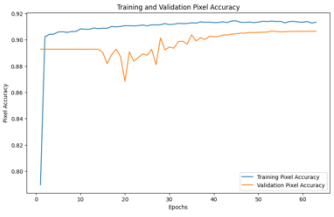
The training process spanned 100 epochs with a batch size of 32. The model demonstrated a good fit, evidenced by the training and validation loss coming together, as well as the progressive improvement across all metrics.

The highest recorded accuracy on the training set was approximately 91.4%, with the validation set achieving around 90.9%. The model's loss function reached its minimum at around 0.2265 for the training set and 0.2451 for the validation set. IoU, a important metric for segmentation tasks, peaked at 0.188 for training and 0.208 for validation, indicating the model's proficiency. F1 score and pixel accuracy further showed the model's robustness, with maximum values of approximately 0.294 & 0.914 for training and 0.319 and 0.906 for validation.

	Metric	Training	Validation
0	Max Accuracy	0.914185	0.909604
1	Min Loss	0.226500	0.245097
2	Max IoU	0.188006	0.208685
3	Max F1 Score	0.294325	0.319384
4	Max Pixel Accuracy	0.914343	0.906649

Training and validation loss

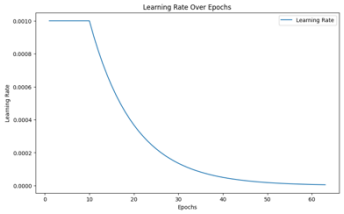
accuracy



Training and validation pixel accuracy

F1 score

IoU score



learning rate

B. U-Net ++

Methodology
We implemented an enhanced architecture, U-Net++, to address complex image segmentation tasks on the Kvasir Instrument dataset. U-Net++ is an advanced model that integrates nested and dense skip pathways with the standard U-Net, aiming to improve the capture of fine-grained details and reduce the semantic gap between the feature maps of the encoder and decoder modules. [1]

Model Architecture

This model uses a series of convolutional blocks in both the encoding and decoding.

```
def conv_block(input_tensor, num_filters):
    x = Conv2D(num_filters, (3, 3), padding='same')(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(num_filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
```

While also using Max pooling like a traditional U-Net model, this is to downsample. On the other hand the model uses an upsample concatenation block to integrate the features from the encoder.

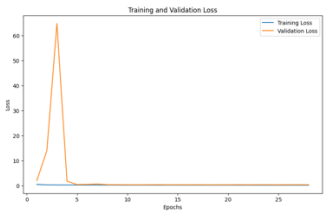
```
def upsample_concat_block(input_tensor, skip_tensor, num_filters):
    x = UpSampling2D((2, 2))(input_tensor)
    x = concatenate([x, skip_tensor])
    x = conv_block(x, num_filters)
    return x
```

Training
The model was compiled with the same metrics of accuracy, loss and early stopping as the U-Net model.

Results
Training was done over 100 epochs with a batch size of 32. The U-Net++ model achievied a maximum training accuracy of approximately 92.8% and a validation accuracy of around 89.6%.

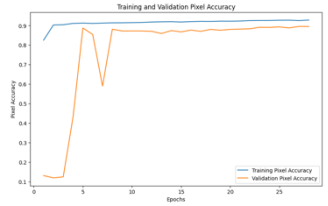
The minimum loss reached was 0.1945 for the training set and 0.3324 for the validation set, indicating efficient learnin. The IoU and F1 score metrics also saw continual improvement, with the training IoU reaching up to 0.2407 and the validation IoU peaking at 0.1131, highlighting the model's effectiveness in segmenting instruments from medical images. Also, the training and validation F1 scores reached a maximum of 0.3638 and 0.1961.

	Metric	Training	Validation
0	Max Accuracy	0.928183	0.895761
1	Min Loss	0.194496	0.332359
2	Max IoU	0.240683	0.113070
3	Max F1 Score	0.363839	0.196068
4	Max Pixel Accuracy	0.928292	0.895720

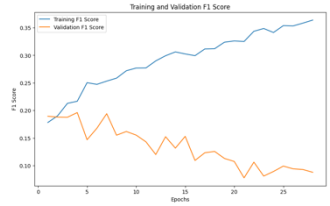


Training and validation loss

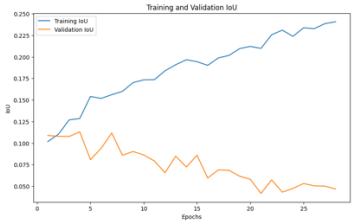
accuracy



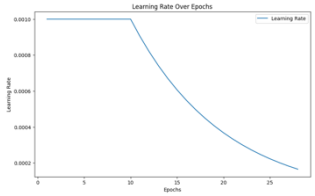
training and validation pixel accuracy



f1 score



Training and val. IoU



learning rate

C. FPN

Methodology
The Feature Pyramid Network (FPN) model is designed for semantic segmentation tasks, it has a pyramidal hierarchy of deep convolutional networks to combine low-resolution, semantically strong features with high-resolution, semantically weak features. The architecture is engineered to enhance feature extraction.[2]

Model
The FPN model uses a bottom-up pathway, a top-down pathway, and lateral connections to extract and merge features across different scales[2]. This means its essentially zooming in and out with different resolutions to better understand the image and in the end perform segmentation based on both bottom-up and top-down results.

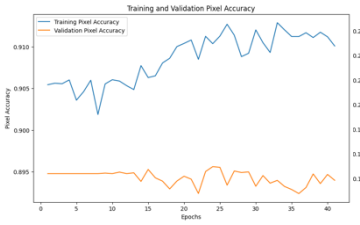
Training
The model was compiled with the same metrics of accuracy, loss and early stopping as the U-Net model and the ++.

Results
Training proceeded for 100 epochs with a batch size of 8. The FPN model showed a steady improvement, achieving a peak training accuracy of 91.17% and a top validation accuracy of 89.56%. The lowest loss was 0.2364 in training and 0.2816 in validation. The IoU metric rised to a maximum of 0.1675 during training and 0.1540 in validation, comfirming the model's segmentation efficacy. The F1 scores also improved and reached a maximum of

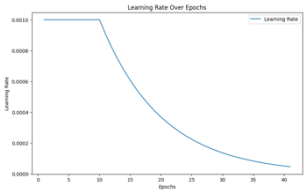
0.2666 in training and 0.2492 in validation. Pixel accuracy metrics showed a high consistency.

	Metric	Training	Validation
0	Max Accuracy	0.911725	0.895563
1	Min Loss	0.236381	0.281555
2	Max IoU	0.167463	0.154031
3	Max F1 Score	0.266627	0.249200
4	Max Pixel Accuracy	0.912887	0.895617

Training and validation loss accuracy



training and validation pixel accuracy f1 score



Training and val. IoU learning rate

III. Three Different Activation Functions (Model)

A Convolutional Neural Network (CNN) with eight layers was developed:

```
Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
MaxPooling2D(2, 2),
Conv2D(64, (3, 3), activation='relu', padding='same'),
MaxPooling2D(2, 2),
Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same'),
Conv2D(32, (3, 3), activation='relu', padding='same'),
Conv2DTranspose(32, (3, 3), strides=(2, 2), padding='same'),
Conv2D(1, (1, 1), activation='sigmoid')
```

Encoders:

1. The initial Conv2D layer, equipped with 32 filters (3x3), extracts features from the input image.

2. A subsequent max pooling layer (2x2) reduces the dimensions of the feature maps, thus mitigating computational load and overfitting.

3. Another Conv2D layer, with 64 filters (3x3), is employed to extract more complex features.

4. This is followed by another max pooling layer, similar to the second layer.

Decoders:

5. Conv2DTranspose layer with 64 filters (3x3) is utilized to increase the dimensions of the feature map, initiating output segmentation map reconstruction.

6. Conv2S layer with 32 filters (3x3) further processes the feature map.

7. Another Conv2DTranspose layer with 32 filters (3x3) mirrors the fifth layer's function.

8. The final layer, a Conv2D with a single 1x1 filter, employs a sigmoid function to generate a probability map, determining pixel class segmentation.

The ReLU activation function is used in the presented model, with variations employing either tanh or sigmoid activation functions. A batch size of 32 and a training duration of 10 epochs were selected for these experiments.

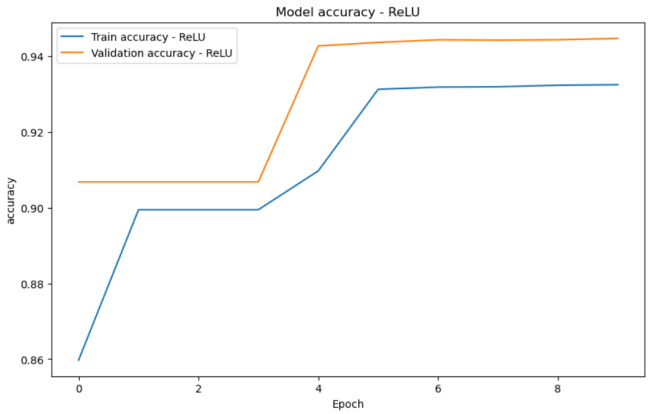
IV. Three Different Activation Functions (Results)

ReLU model:

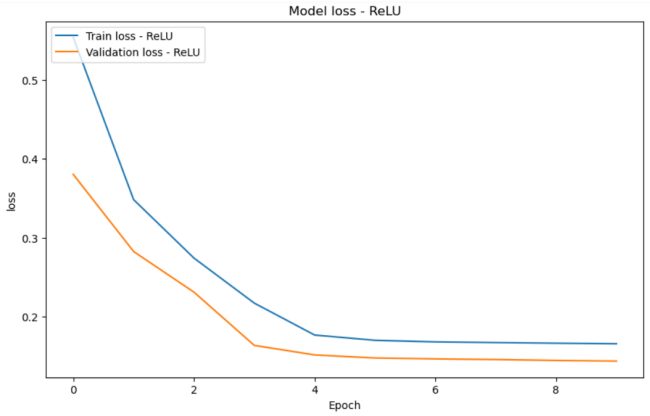
```
Epoch 1/10
15/15 [==============================] - 43s 3s/step - loss: 0.5540 - accuracy: 0.8597 - val_loss: 0.3803 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 42s 3s/step - loss: 0.3484 - accuracy: 0.8995 - val_loss: 0.2826 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 39s 3s/step - loss: 0.2743 - accuracy: 0.8995 - val_loss: 0.2310 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 40s 3s/step - loss: 0.2171 - accuracy: 0.8995 - val_loss: 0.1635 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 39s 3s/step - loss: 0.1766 - accuracy: 0.9097 - val_loss: 0.1515 - val_accuracy: 0.942
8 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 40s 3s/step - loss: 0.1700 - accuracy: 0.9313 - val_loss: 0.1476 - val_accuracy: 0.943
7 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 40s 3s/step - loss: 0.1680 - accuracy: 0.9319 - val_loss: 0.1464 - val_accuracy: 0.944
4 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 40s 3s/step - loss: 0.1671 - accuracy: 0.9320 - val_loss: 0.1456 - val_accuracy: 0.944
3 - lr: 1.0000e-04
Epoch 9/10
15/15 [==============================] - 37s 2s/step - loss: 0.1663 - accuracy: 0.9324 - val_loss: 0.1444 - val_accuracy: 0.944
4 - lr: 1.0000e-04
Epoch 10/10
15/15 [==============================] - 37s 2s/step - loss: 0.1656 - accuracy: 0.9325 - val_loss: 0.1436 - val_accuracy: 0.944
8 - lr: 1.0000e-04
```

The ReLU model achieved an accuracy of 0.9325 and a loss of 0.1656 in its final epoch, with corresponding validation accuracy and loss values of 0.944 and 0.1436, respectively.

Graph for accuracy:



Graph for loss:

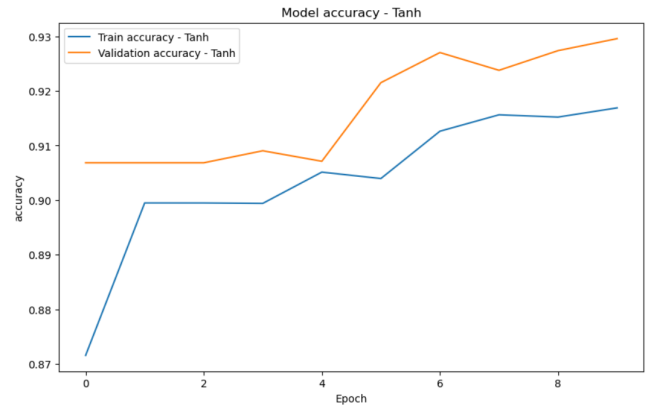


Tahn model:

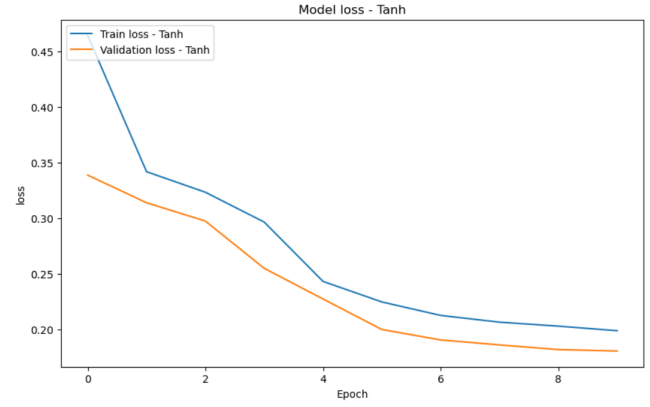
```
Epoch 1/10
15/15 [==============================] - 42s 3s/step - loss: 0.4644 - accuracy: 0.8716 - val_loss: 0.3387 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 39s 3s/step - loss: 0.3418 - accuracy: 0.8995 - val_loss: 0.3139 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 38s 3s/step - loss: 0.3233 - accuracy: 0.8995 - val_loss: 0.2974 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 38s 3s/step - loss: 0.2965 - accuracy: 0.8994 - val_loss: 0.2548 - val_accuracy: 0.909
0 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 38s 3s/step - loss: 0.2431 - accuracy: 0.9051 - val_loss: 0.2272 - val_accuracy: 0.907
1 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 38s 3s/step - loss: 0.2246 - accuracy: 0.9040 - val_loss: 0.1998 - val_accuracy: 0.921
5 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 38s 3s/step - loss: 0.2125 - accuracy: 0.9126 - val_loss: 0.1904 - val_accuracy: 0.927
0 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 38s 3s/step - loss: 0.2064 - accuracy: 0.9156 - val_loss: 0.1859 - val_accuracy: 0.923
8 - lr: 1.0000e-04
Epoch 9/10
15/15 [==============================] - 39s 3s/step - loss: 0.2028 - accuracy: 0.9152 - val_loss: 0.1817 - val_accuracy: 0.927
4 - lr: 1.0000e-04
Epoch 10/10
15/15 [==============================] - 37s 3s/step - loss: 0.1987 - accuracy: 0.9169 - val_loss: 0.1804 - val_accuracy: 0.929
6 - lr: 1.0000e-04
```

The Tanh model recorded an accuracy of 0.9169 and a loss of 0.1987 in the final epoch. Validation accuracy and loss were observed at 0.929 and 0.1904, respectively.

Graph for accuracy:



Graph for loss:

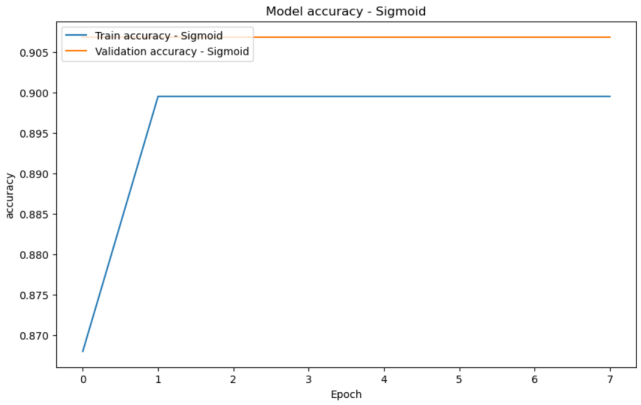


Sigmoid model:

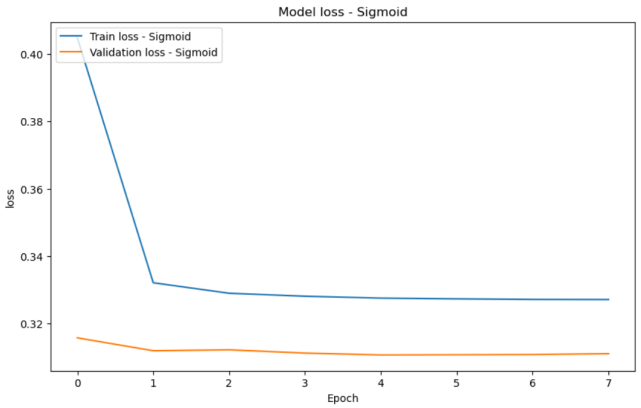
```
Epoch 1/10
15/15 [==============================] - 41s 3s/step - loss: 0.4047 - accuracy: 0.8680 - val_loss: 0.3158 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 39s 3s/step - loss: 0.3321 - accuracy: 0.8995 - val_loss: 0.3120 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 38s 3s/step - loss: 0.3290 - accuracy: 0.8995 - val_loss: 0.3123 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 39s 3s/step - loss: 0.3281 - accuracy: 0.8995 - val_loss: 0.3113 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 38s 3s/step - loss: 0.3276 - accuracy: 0.8995 - val_loss: 0.3107 - val_accuracy: 0.906
8 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 39s 3s/step - loss: 0.3274 - accuracy: 0.8995 - val_loss: 0.3108 - val_accuracy: 0.906
8 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 41s 3s/step - loss: 0.3272 - accuracy: 0.8995 - val_loss: 0.3109 - val_accuracy: 0.906
8 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 41s 3s/step - loss: 0.3272 - accuracy: 0.8995 - val_loss: 0.3111 - val_accuracy: 0.906
8 - lr: 1.0000e-04
```

The Sigmoid model maintained a consistent accuracy of 0.8995 over the last nine epochs, with a loss of 0.3272 in the two last epochs. Validation accuracy remained steady at 0.906, with the lowest observed loss at 0.3107. Because the validation loss did not improve, the early stopping method stopped the model after epoch 8.

Graph for accuracy:



Graph for loss:



V. THREE DIFFERENT OPTIMIZATION METHODS (RESULTS)

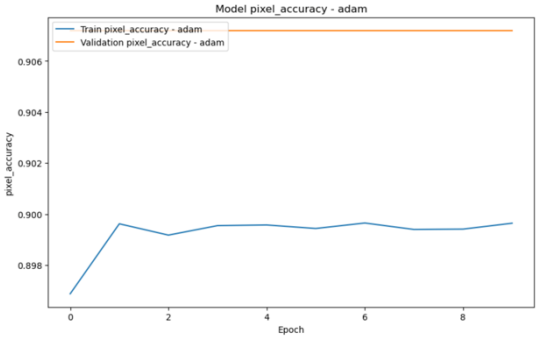
For this part of the task we used the same models, but integrated it with the optimization methods. The Adam, Adagrad, and SGD (Stochastic Gradient Descent) optimizers are all popular algorithms used in training neural networks and these are the three methods we chose for this task. We also implemented pixel accuracy for a better view of the results.

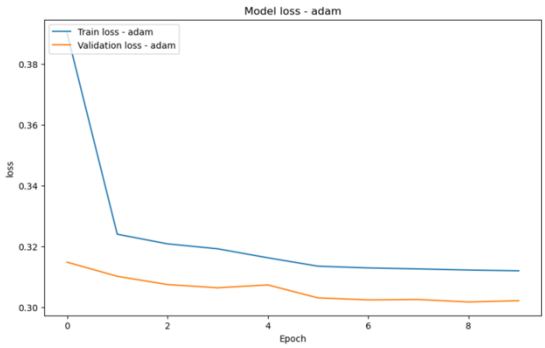
For this task (image segmentation) the adam optimizer is the preferd method due to its efficiency in handling sparse gradients and its adaptability. Adagrad and SDG can be used, but it is a bit trickier to tune them into being as accurate as the Adam optimizer but is most of the time less effective for image segmentation.

Adam optimizer:

```
Epoch 1/10
15/15 [==============================] - 59s 4s/step - loss: 0.3842 - accuracy: 0.8902 - pixel_accuracy: 0.8903 - v
al_loss: 0.3128 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 67s 5s/step - loss: 0.3229 - accuracy: 0.8995 - pixel_accuracy: 0.8994 - v
al_loss: 0.3098 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 71s 5s/step - loss: 0.3203 - accuracy: 0.8995 - pixel_accuracy: 0.8992 - v
al_loss: 0.3085 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 72s 5s/step - loss: 0.3188 - accuracy: 0.8995 - pixel_accuracy: 0.8992 - v
al_loss: 0.3060 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 74s 5s/step - loss: 0.3177 - accuracy: 0.8995 - pixel_accuracy: 0.8994 - v
al_loss: 0.3056 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 75s 5s/step - loss: 0.3161 - accuracy: 0.8995 - pixel_accuracy: 0.8999 - v
al_loss: 0.3054 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 77s 5s/step - loss: 0.3158 - accuracy: 0.8995 - pixel_accuracy: 0.8994 - v
al_loss: 0.3048 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 77s 5s/step - loss: 0.3156 - accuracy: 0.8995 - pixel_accuracy: 0.8993 - v
al_loss: 0.3045 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 9/10
15/15 [==============================] - 79s 5s/step - loss: 0.3154 - accuracy: 0.8995 - pixel_accuracy: 0.8996 - v
al_loss: 0.3042 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 10/10
15/15 [==============================] - 79s 5s/step - loss: 0.3152 - accuracy: 0.8995 - pixel_accuracy: 0.8997 - v
al_loss: 0.3046 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
```

Adam optimizer graphs:

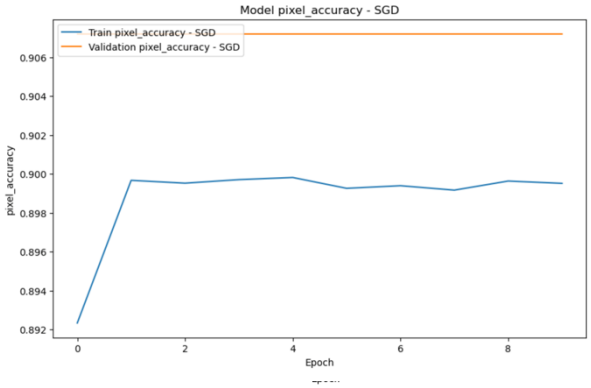


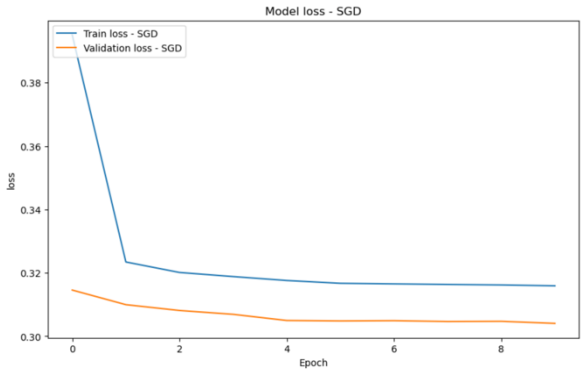


SDG optimizer:

```
Epoch 1/10
15/15 [==============================] - 87s 6s/step - loss: 0.3979 - accuracy: 0.8959 - pixel_accuracy: 0.8960 - v
al_loss: 0.3108 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 81s 5s/step - loss: 0.3231 - accuracy: 0.8995 - pixel_accuracy: 0.8994 - v
al_loss: 0.3095 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 81s 5s/step - loss: 0.3203 - accuracy: 0.8995 - pixel_accuracy: 0.8998 - v
al_loss: 0.3086 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 83s 6s/step - loss: 0.3189 - accuracy: 0.8995 - pixel_accuracy: 0.8996 - v
al_loss: 0.3068 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 84s 6s/step - loss: 0.3169 - accuracy: 0.8995 - pixel_accuracy: 0.8993 - v
al_loss: 0.3056 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 85s 6s/step - loss: 0.3154 - accuracy: 0.8995 - pixel_accuracy: 0.8994 - v
al_loss: 0.3056 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 148s 10s/step - loss: 0.3151 - accuracy: 0.8995 - pixel_accuracy: 0.8995 -
val_loss: 0.3050 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 205s 14s/step - loss: 0.3149 - accuracy: 0.8995 - pixel_accuracy: 0.8993 -
val_loss: 0.3047 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 9/10
15/15 [==============================] - 103s 7s/step - loss: 0.3146 - accuracy: 0.8995 - pixel_accuracy: 0.8996 -
val_loss: 0.3047 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 10/10
15/15 [==============================] - 90s 6s/step - loss: 0.3145 - accuracy: 0.8995 - pixel_accuracy: 0.8992 - v
al_loss: 0.3040 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
```

SDG optmizer grapghs:

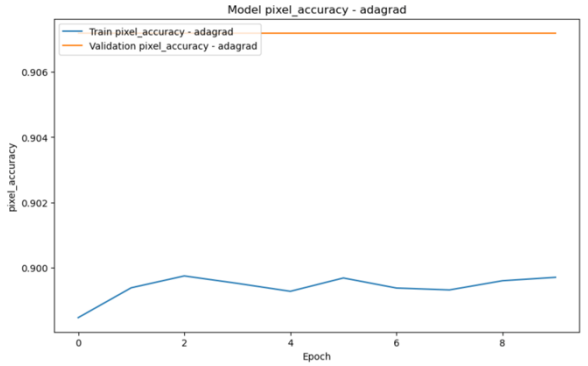


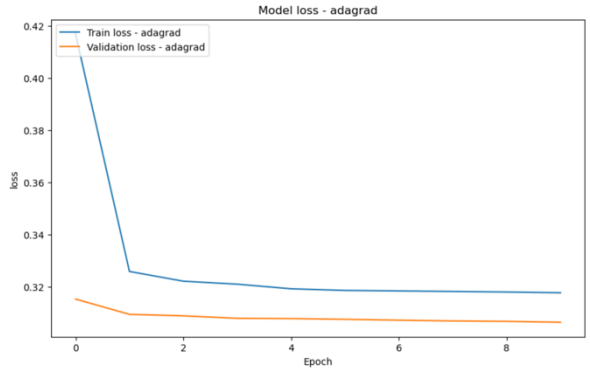


Adagrad optimizer:

```
Epoch 1/10
15/15 [==============================] - 95s 6s/step - loss: 0.4498 - accuracy: 0.8983 - pixel_accuracy: 0.8983 - v
al_loss: 0.3150 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 2/10
15/15 [==============================] - 84s 6s/step - loss: 0.3257 - accuracy: 0.8995 - pixel_accuracy: 0.8996 - v
al_loss: 0.3106 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 3/10
15/15 [==============================] - 91s 6s/step - loss: 0.3230 - accuracy: 0.8995 - pixel_accuracy: 0.8996 - v
al_loss: 0.3087 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 4/10
15/15 [==============================] - 74s 5s/step - loss: 0.3214 - accuracy: 0.8995 - pixel_accuracy: 0.8992 - v
al_loss: 0.3079 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 5/10
15/15 [==============================] - 69s 5s/step - loss: 0.3194 - accuracy: 0.8995 - pixel_accuracy: 0.8997 - v
al_loss: 0.3055 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 0.0010
Epoch 6/10
15/15 [==============================] - 70s 5s/step - loss: 0.3177 - accuracy: 0.8995 - pixel_accuracy: 0.8995 - v
al_loss: 0.3053 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 7/10
15/15 [==============================] - 74s 5s/step - loss: 0.3171 - accuracy: 0.8995 - pixel_accuracy: 0.8995 - v
al_loss: 0.3052 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 8/10
15/15 [==============================] - 69s 5s/step - loss: 0.3165 - accuracy: 0.8995 - pixel_accuracy: 0.8993 - v
al_loss: 0.3049 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 9/10
15/15 [==============================] - 85s 6s/step - loss: 0.3159 - accuracy: 0.8995 - pixel_accuracy: 0.8990 - v
al_loss: 0.3044 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
Epoch 10/10
15/15 [==============================] - 70s 5s/step - loss: 0.3155 - accuracy: 0.8995 - pixel_accuracy: 0.8995 - v
al_loss: 0.3041 - val_accuracy: 0.9068 - val_pixel_accuracy: 0.9072 - lr: 1.0000e-04
```

Adagrad optimizer graphs:





Even though the accuracy is almost the same in this case, we see that the pixel accuracy for adam optimizer is the highest one as predicted for this kind of task.

VI. Conclusion

We explored a number of different methods and functions and it
was really exciting to see how each one performed compared to each other, and how combining different strategies to improve loss and accuracy metrics of a model worked. It all comes down to the dataset and the task at hand. The kvasir instrument dataset was an exciting dataset for us to implement image segmentation on because of its real-world likeness.

[1] https://medium.com/@saba99/unet-443b429ae0be

[2] https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

[2]