Painter prediction

Creating environment

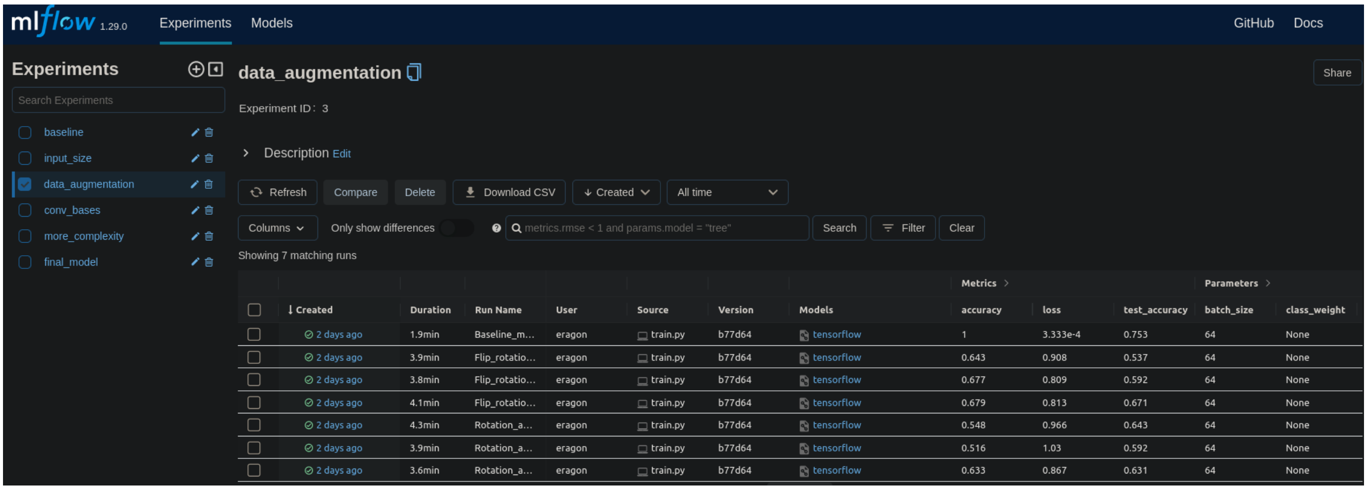
Tools used

- MLFlow
- Conda

Conda

- Conda is used to create the environment and make sure it works across devices

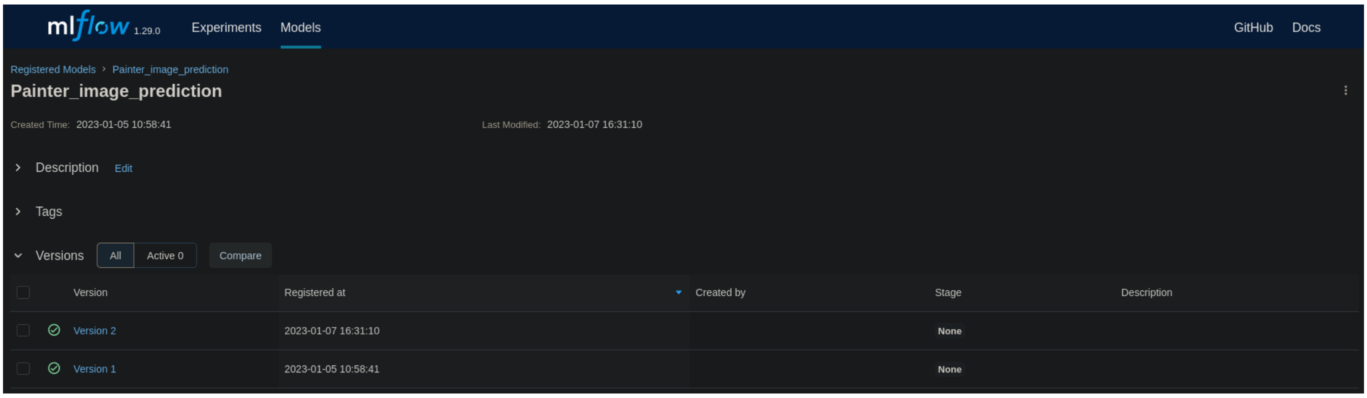
MLflow



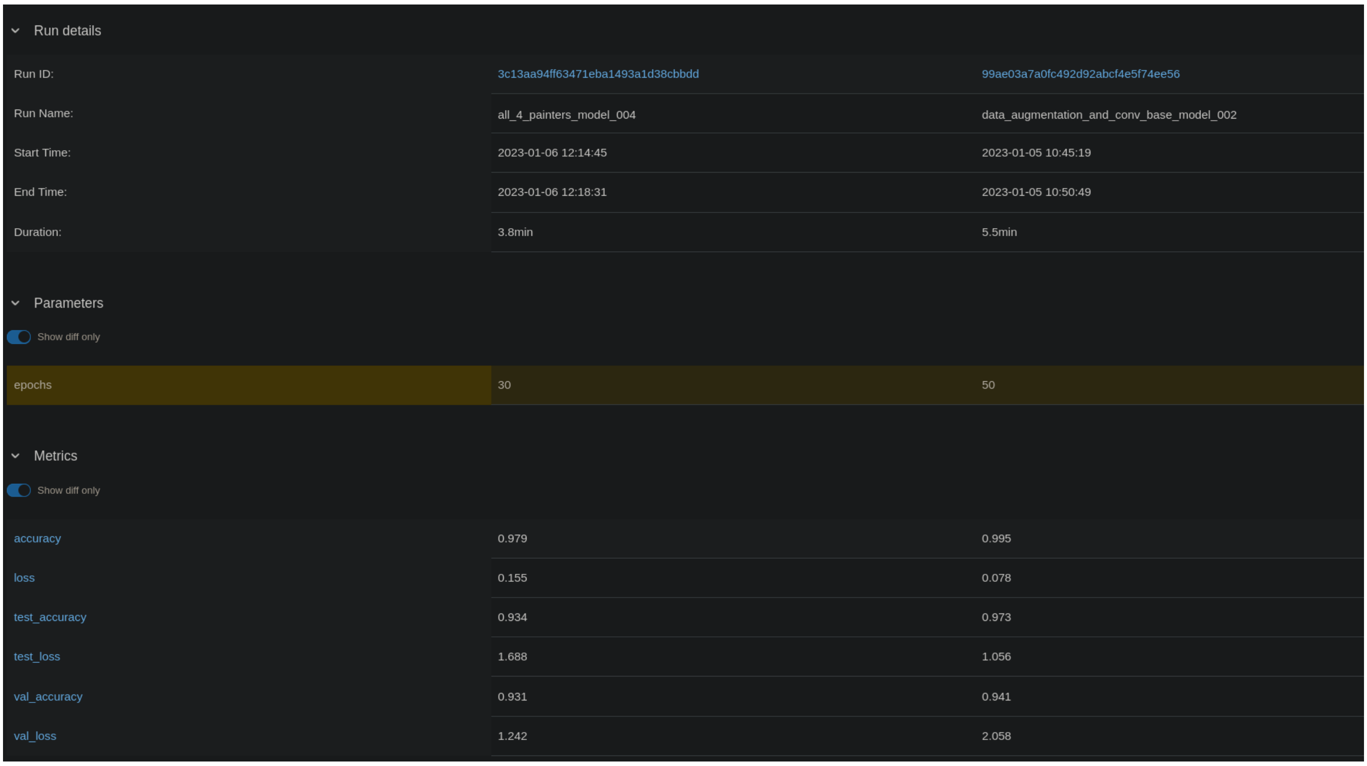
Why mlflow

Mlflow was used to manage the models and keep note of what changes were made to the model and what effect that had on the metrics/result of the model. In mlflow we have experiments. These are used to organize runs in categories.

Mlflow also give the option to host the models. This can be used to manage what model version is active at each moment.

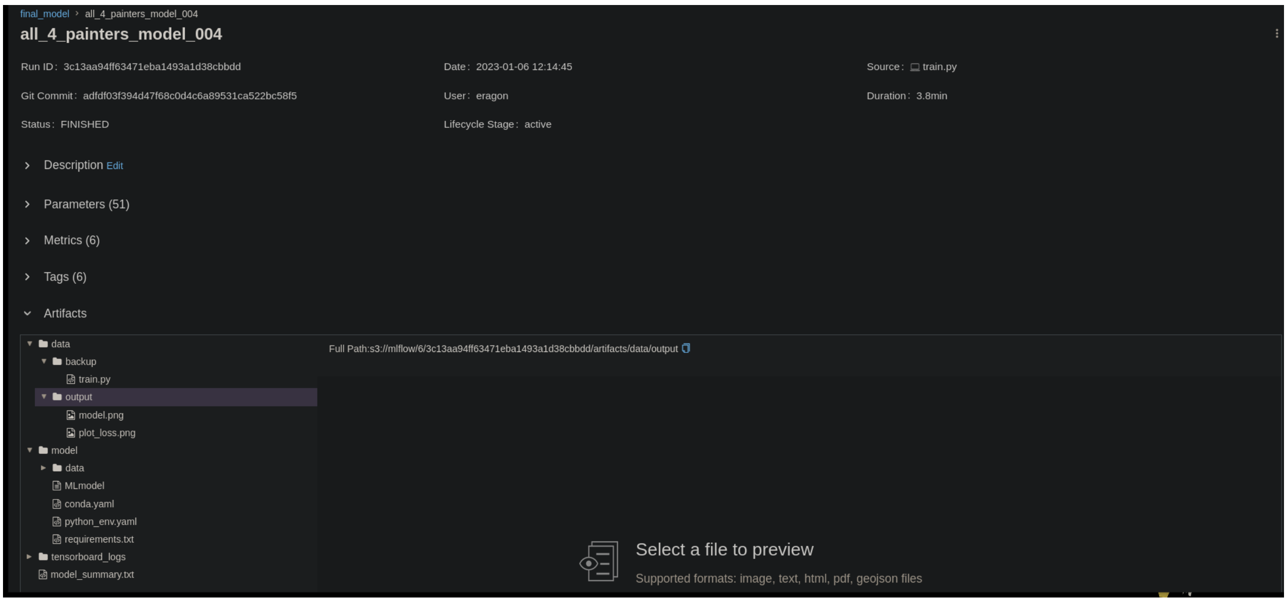


Mlflow was also used to compare models agains each other.



Artifacts

- Mlflow also logs the artifacts of each run



- With mlflow_helpers and the class `Mlflow_controller`

    - the `train.py` is logged to mlflow
        - This makes it possible to see exact what code is used to make the model
        - And removes the chances of losing what change affected the metrics
    - With the artifacts there are also all files required to deploy the model on another server
        - Using conda or virtual environments

Deploying mlflow

- The docker compose file was used to deploy mlflow server
    - Because this also deployes Minio and the SQL database
        - Minio and the database are used to manage the artifacts and the models (the deployed models)
    - Normaly to deploy an model from mlflow `mlflow models serve` command is used
        - Here the website downloaded the model and predicts it.
        - The website should normaly send the data to the server that runs the model
        - But that was outside of the material seen in the class
- For ease of use mlflow with all the trained models has been deployed on an home server
    - Ask the creator for access
    - All models can be downloaded from the mlflow server
        - An example of this can be found in website.py

mlflow_Controller

- This class was created to manage the runs in mlflow
- Sequence of the used classes

1. load_features()
2. _set_train_options()
3. _build_model()
4. train()
5. mlflow_log()

- The class has been documented for more explanation on the working
- This simplified the used of mlflow and the train.py file was cleaner and easier to understand

Scraping

- The scraping can be found in the testing.ipynb
- To scrape paintings for Rembrandt the website rembrandtpainting.net was used
    - Wheb exploring the website, the page complete catalogue had an lage collection of paintings easy to download
    - 214 images were collected here

Preprocessing

- This was made to create the datasets for training, validation and test.
- Takes an random sample from the images and resizes
    - Then places the new datasets in `data/preprocessed`

- The training only used this folder and doesn't touch the `data/raw` folder

Baseline

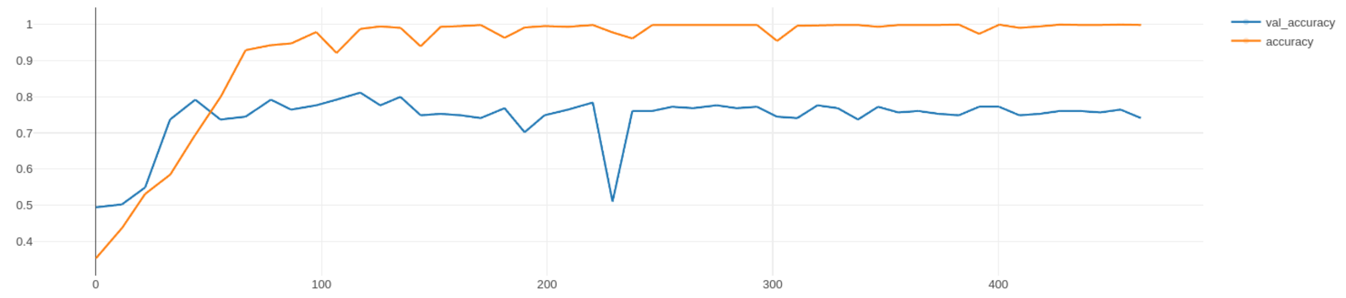
The baseline should be $1/(number_of_painters)$.

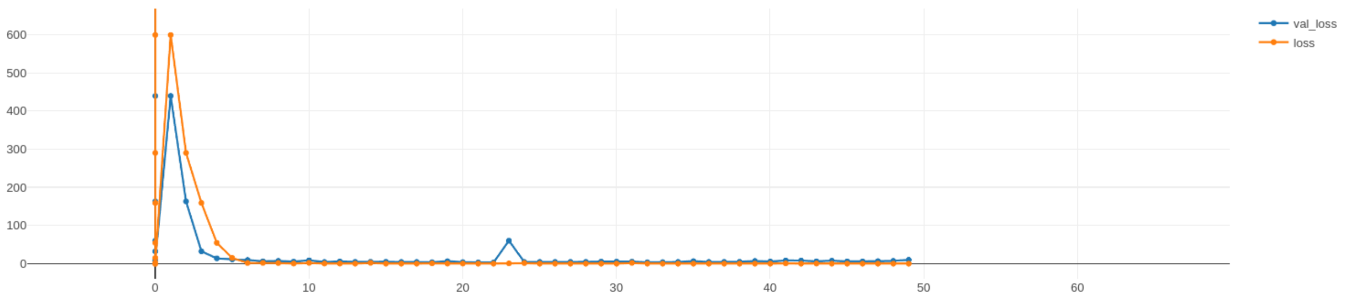
But to have an more accurate baseline, a simple one layer model was created. This was also used to test/create the environment with mlflow.

- Model

```
inputs = keras.Input(shape=input_shape)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.5)(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
outputs = layers.Dense(output_shape, activation="softmax")(x)
```

- Metrics logged with mlflow





Input size

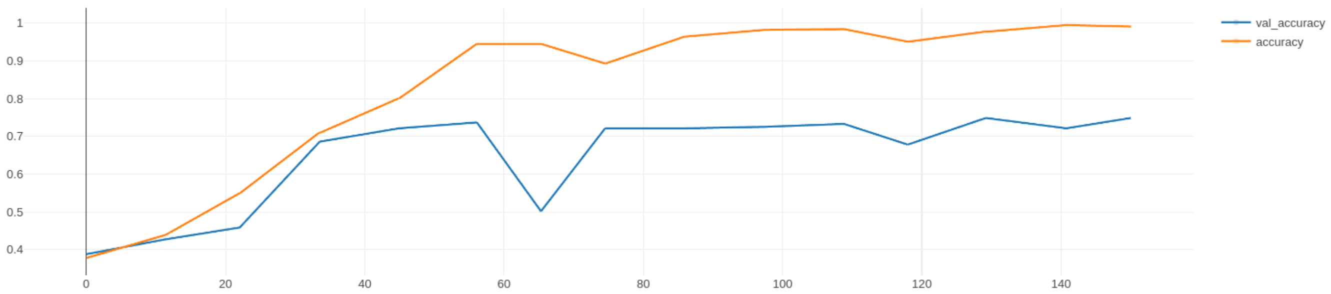
Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Baseline	0.991	0.749	0.733	6.7
Increased image size	0.929	0.71	0.725	13.3
Decreased image size	0.385	0.392	0.392	13.2

- We will use 180 x 180 shape for now
  - For the faster training speed
  - But will check if we added some more complexity to the model what the image shape will do then

180 x 180 (baseline)

- This is the baseline for this example.
- 15 epochs to speed up the testing



400 x 400 (increased image size)

- Increased image size
- Takes a lot longer to train

input size baseline

60 x 50 (decreased image size)

- Increased image size
- We can see it doesn't train
- We need to reduce the Trainable params or increase the input shape

input size baseline

Data augmentation

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Baseline	1	0.69	0.69	1.9
Rotation_and_zoom (0.1, 0.2)	0.633	0.604	0.631	3.6
Rotation_and_zoom (0.9, 0.9)	0.516	0.533	0.592	3.9
Rotation_and_zoom (0.5, 0.5)	0.548	0.533	0.643	4.3
Flip, rotation_and_zoom (horizontal, 0.1, 0.2)	0.679	0.608	0.671	4.1
Flip, rotation_and_zoom (vertical, 0.1, 0.2)	0.677	0.588	0.592	3.8

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Flip, rotation_and_zoom (horizontal_and_vertical, 0.1, 0.2)	0.643	0.482	0.537	3.9

Baseline

- 50 epoches were used to give the model a better chance to against data augmentation
- Some changes were made to the environment so the times will be a lot different between this experiment and the previous experiments.
  - These changes can be found in problem solving

Rotation and zoom

- Rotation and zoom were added to the model.
- The training accuracy is a lot lower then the baseline but the Val and test accuracy is in the same line as the train
  - So definitely no overfitting
  - Also the model not good enough
    - Other changes need to be tested with data augmentation active
    - So the improvements will be better visible

Flip, rotation and zoom

- Flip, rotation and zoom were added to the model.
- Again there were no improvements here compaired to the baseline.
  - But Flip, rotation_and_zoom (horizontal, 0.1, 0.2) is the best data augmentation so this will be used going further

Conv bases

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Baseline	1	0.737	0.78	1.7
vgg16	0.999	0.922	0.969	3.2
vgg19	1	0.957	0.988	4.3
ResNet152	0.998	0.933	0.973	5.3
ResNet152_V2	0.998	0.937	0.969	5.1
Xception	0.999	0.894	0.949	2.4

vgg16

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
conv base fully trainable	0.379	0.392	0.392	9.2
conv base not trainable	0.999	0.925	0.965	3.1
conv base last 4 layers trainable	0.387	0.392	0.392	3.3
conv base last 2 layers trainable	0.999	0.922	0.969	3.2

Difference

- The first model is fully trainable
  - It fails because it has to much parameters it can train
  - Takes an long time to train
- The second model is the conv base not trainable
  - This gives an better prediction then the base model
  - Relative fast to train
- 4 trainable layers also doesn't learn
  - To much parameters again
- 2 trainable layers works best
  - Better then no trainable layers
  - Doesn't take to much time to train

vgg19

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
conv base not trainable	1	0.922	0.957	4.5
conv base last 4 layers trainable	0.389	0.392	0.392	4.3
conv base last 2 layers trainable	1	0.957	0.988	4.3

Difference

- The model with 2 trainable layers gives very good results
- 4 trainable layers doesn't train

RESNET

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
ResNet101 not trainable	0.997	0.937	0.969	3.6
ResNet101 last 4 layers trainable	1	0.91	0.949	3.5
ResNet101 last 2 layers trainable	0.997	0.933	0.961	3.6

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
ResNet152 last 4 layers trainable	0.998	0.933	0.973	5.3
ResNet152 last 2 layers trainable	1	0.937	0.957	5.4
ResNet50 last 4 layers trainable	0.997	0.933	0.969	2.6
ResNet50 last 2 layers trainable	0.998	0.914	0.949	2.7

Difference

- ResNet101 has very nice results
  - no trainable works best
  - but 2 trainable layers is close (compared on val accuracy and test accuracy)
- ResNet152 also works very well
  - 4 trainable layers is the best
- ResNet152 also works very well
  - 4 trainable layers is the best
- ResNet152 4 trainable layers is the best model over validation and test accuracy

RESNET_v2

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
ResNet152V2 not trainable	0.998	0.937	0.969	5.1
ResNet152V2 last 4 layers trainable	0.998	0.922	0.949	4.6
ResNet152V2 last 2 layers trainable	0.998	0.914	0.949	5.5

Difference

- This also has nice results but ResNet152 v1 performes betteru

xception

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
xception last 4 layers trainable	0.999	0.882	0.941	2.5
xception last 2 layers trainable	0.999	0.894	0.949	2.4

Difference

- Here the model with 2 trainable layers performes better

More complexity

Summary

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Baseline	0.999	0.929	0.965	3.6

Baseline

- best model of conv base

```
conv_base = keras.applications.vgg19.VGG19(
            weights="imagenet",
            include_top=False
            )

conv_base.trainable = True
for layer in conv_base.layers[:-2]:
    layer.trainable = False


inputs = keras.Input(shape=input_shape)
x = inputs
x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(output_shape, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```

Other models tried

- This adds more layers on top of the baseline
-

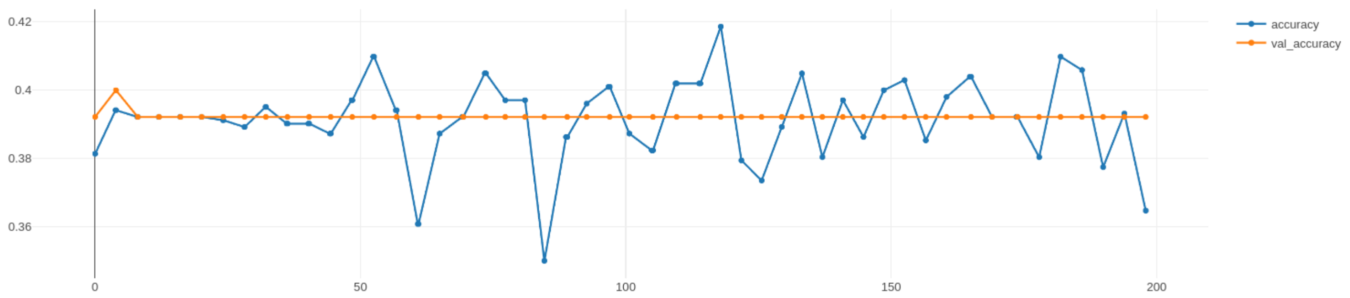
Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Conv_layers_model_001	0.365	0.392	0.392	3.8
Concatinate_model_001	1	0.706	0.725	3.4
Concatinate_model_002	1	0.765	0.78	3.4
Concatinate_model_003	0.976	0.231	0.227	3.0

Conv2D layers model 1

- This model added 1 extra layer after the conv base

- This doesn't learn anymore

```
conv_base = keras.applications.vgg19.VGG19(
            weights="imagenet",
            include_top=False
            )

conv_base.trainable = True
for layer in conv_base.layers[:-2]:
    layer.trainable = False


inputs = keras.Input(shape=input_shape)
x = inputs
x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(output_shape, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```

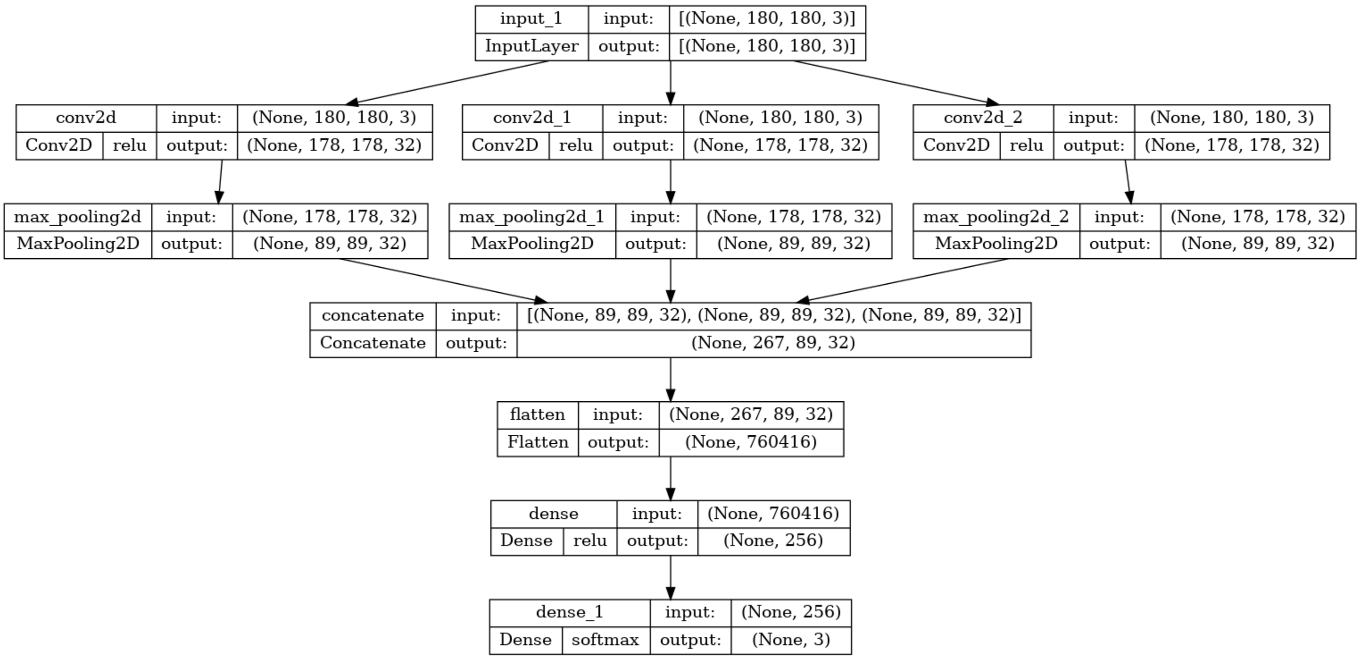


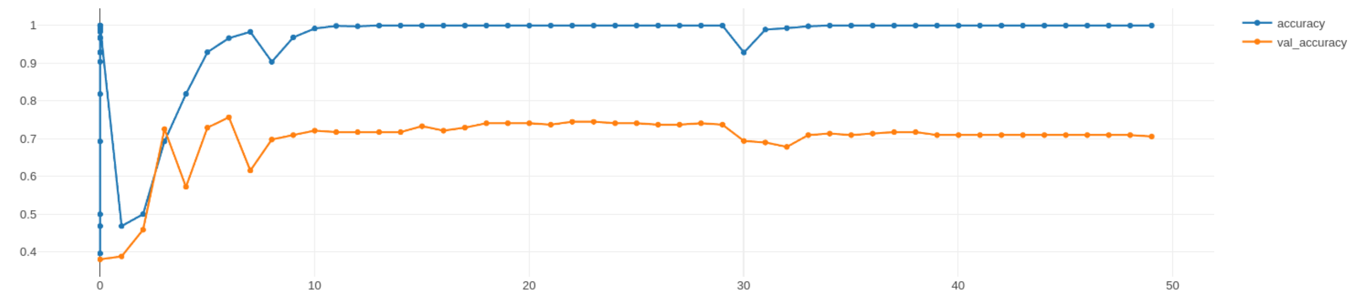
Concatinate model 001

- Tried to see if concatinating models would work
- Results were a bit better then the first baseline model but a lot worse than any conv base that trained

```
inputs = keras.Input(shape=input_shape)
tower_1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")
(inputs)
tower_1 = layers.MaxPooling2D(pool_size=2)(tower_1)

tower_2 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")
```

```
(inputs)
tower_2 = layers.MaxPooling2D(pool_size=2)(tower_2)

tower_3 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")
(inputs)
tower_3 = layers.MaxPooling2D(pool_size=2)(tower_3)


x = keras.layers.concatenate([tower_1, tower_2, tower_3], axis=1)
x = layers.Flatten()(x)

x = layers.Dense(256, activation='relu')(x)

outputs = layers.Dense(output_shape, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```

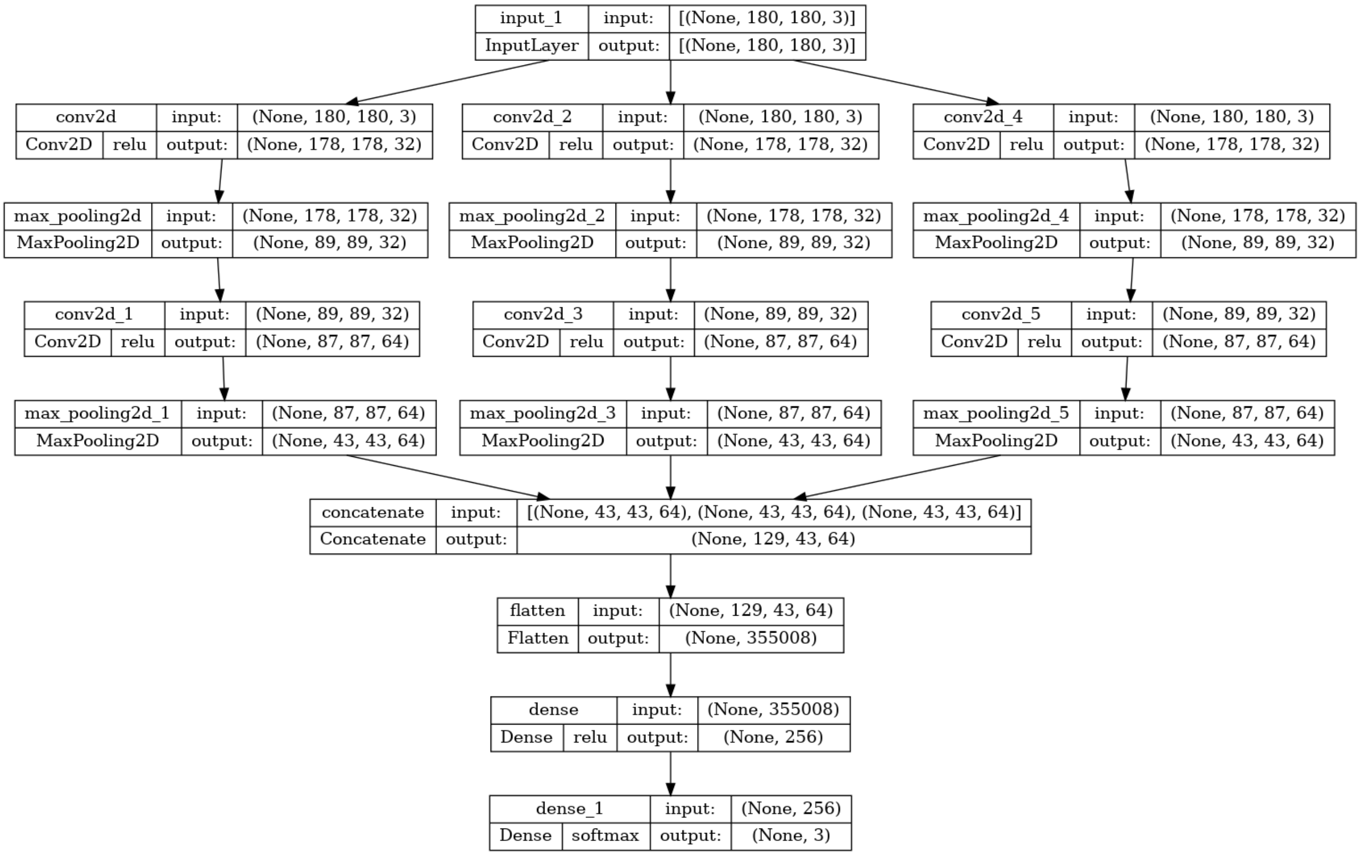


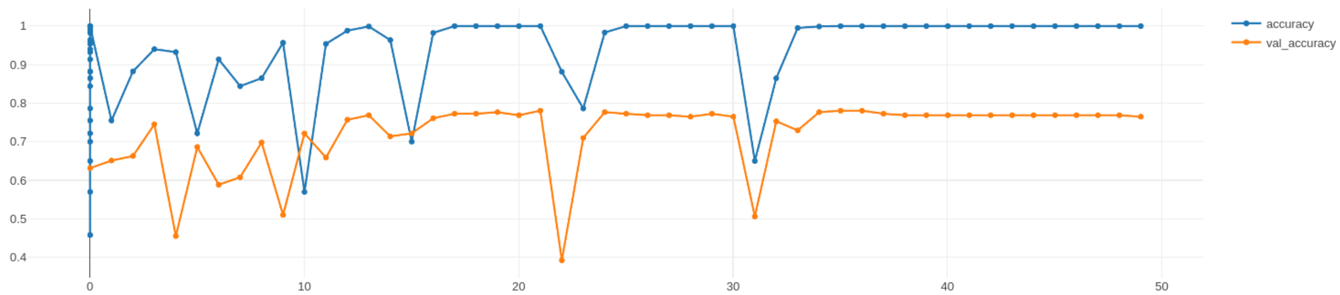


Concatinate model 002

- This added another layer to the towers
- Results were a better than the first Concatinate model

```
inputs = keras.Input(shape=input_shape)
towers = []
num_of_towers = 3
for i in range(num_of_towers):

    tower = layers.Conv2D(filters=32, kernel_size=3, activation="relu")
(inputs)
    tower = layers.MaxPooling2D(pool_size=2)(tower)
    tower = layers.Conv2D(filters=64, kernel_size=3, activation="relu")
(tower)
    tower = layers.MaxPooling2D(pool_size=2)(tower)

    towers.append(tower)


x = keras.layers.concatenate(towers, axis=1)
x = layers.Flatten()(x)

x = layers.Dense(256, activation='relu')(x)

outputs = layers.Dense(output_shape, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```



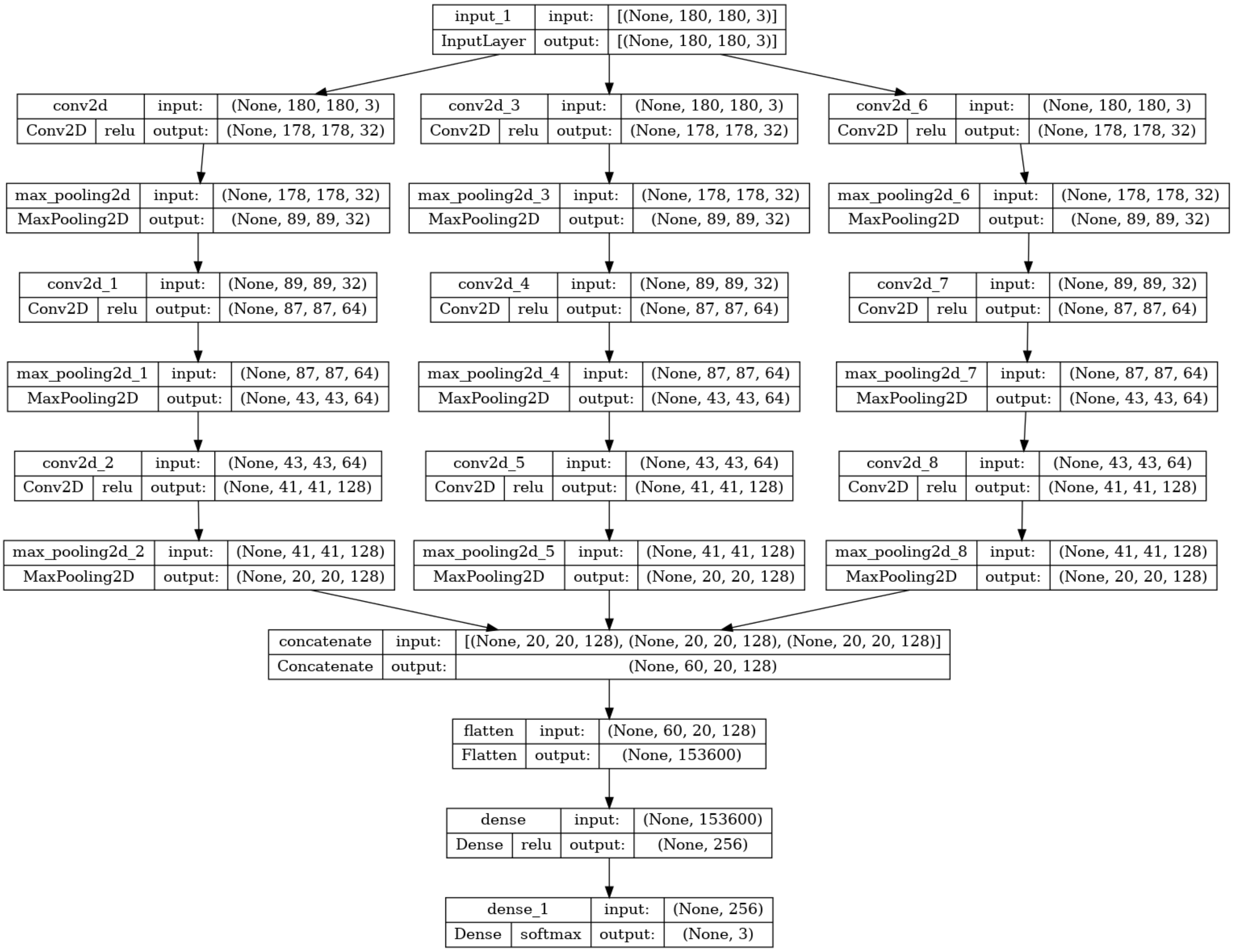


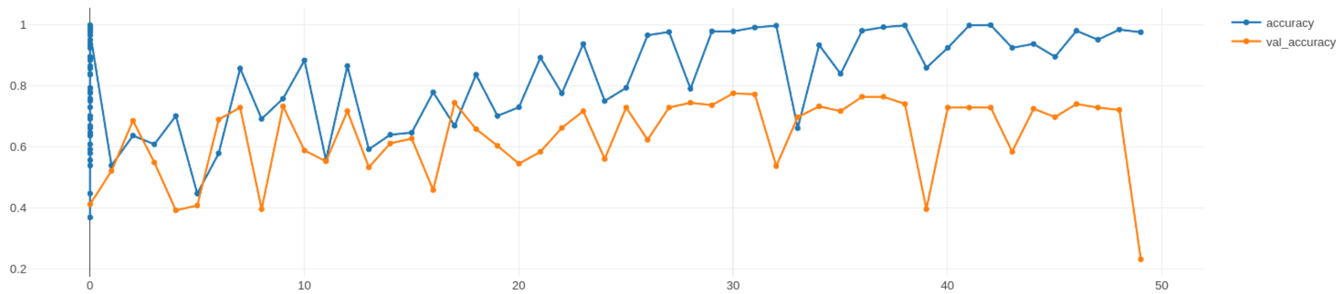
Concatinate model 003

- This added another layer to the towers
- Results were a better than the first Concatinate model

```
inputs = keras.Input(shape=input_shape)
towers = []
num_of_towers = 3
for i in range(num_of_towers):

    tower = layers.Conv2D(filters=32, kernel_size=3, activation="relu")
(inputs)
    tower = layers.MaxPooling2D(pool_size=2)(tower)
    tower = layers.Conv2D(filters=64, kernel_size=3, activation="relu")
(tower)
    tower = layers.MaxPooling2D(pool_size=2)(tower)
    tower = layers.Conv2D(filters=128, kernel_size=3, activation="relu")
(tower)
    tower = layers.MaxPooling2D(pool_size=2)(tower)
```

```
        towers.append(tower)


    x = keras.layers.concatenate(towers, axis=1)
    x = layers.Flatten()(x)

    x = layers.Dense(256, activation='relu')(x)

    outputs = layers.Dense(output_shape, activation="softmax")(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
    ["accuracy"])
```



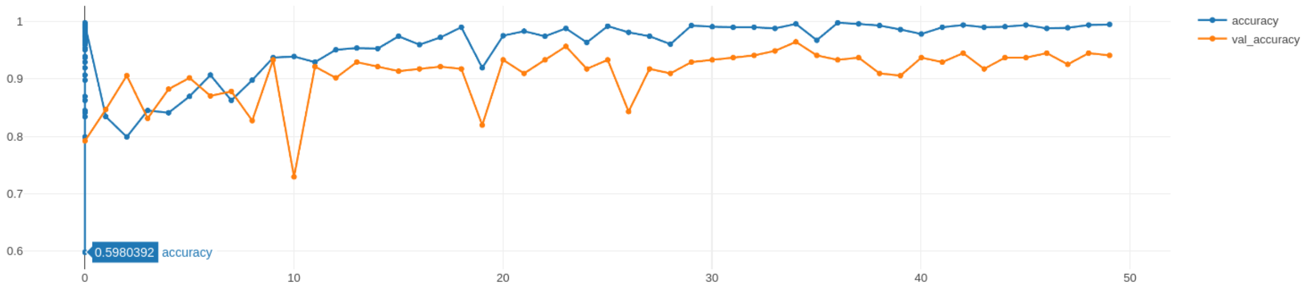


Final model

3 painters

- The final models uses
  - 180 x 180 as image size
  - Flip, rotation_and_zoom (horizontal, 0.1, 0.2)
  - VGG19 2 trainable layers as conv base final_model_3_painters_accuracy this gave the best metrics.

Model	Train_acc	Val_acc	Test_acc	Training Time (in minutes)
Final model with 3 painters	0.995	0.941	0.973	5.5
Final model with 4 painters	0.979	0.931	0.934	3.8

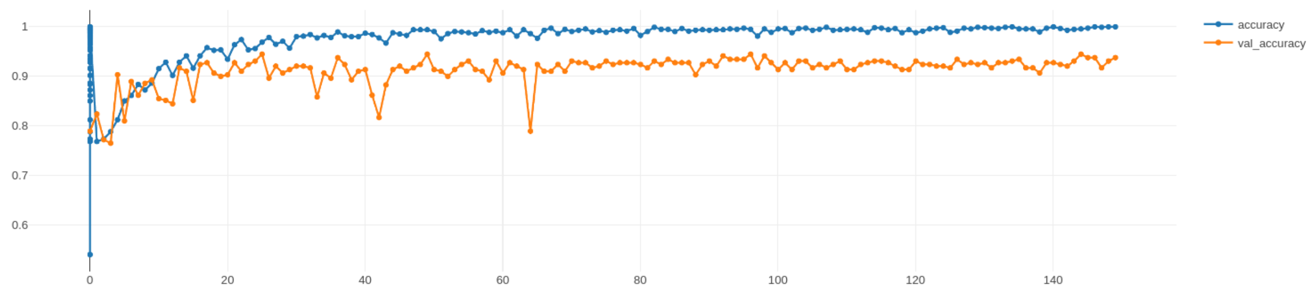



```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.2),
    ])

conv_base = keras.applications.vgg19.VGG19(
            weights="imagenet",
            include_top=False
            )

conv_base.trainable = True
for layer in conv_base.layers[:-2]:
    layer.trainable = False


inputs = keras.Input(shape=input_shape)
x = inputs
x = data_augmentation(x)

x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(output_shape, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

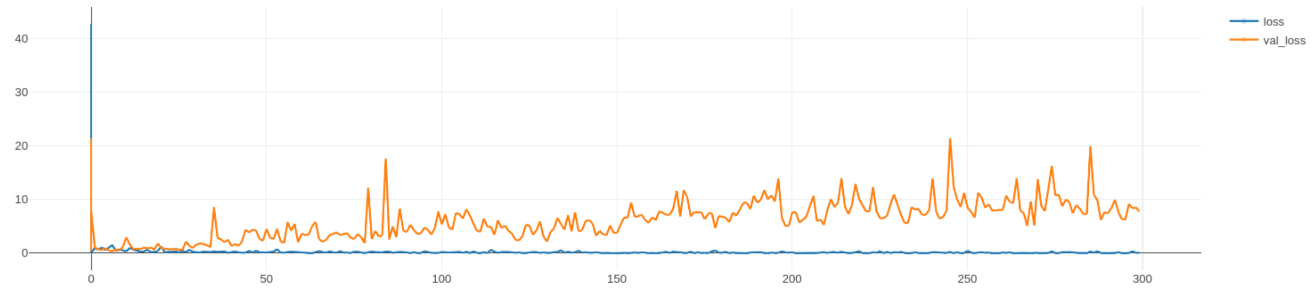
```
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```

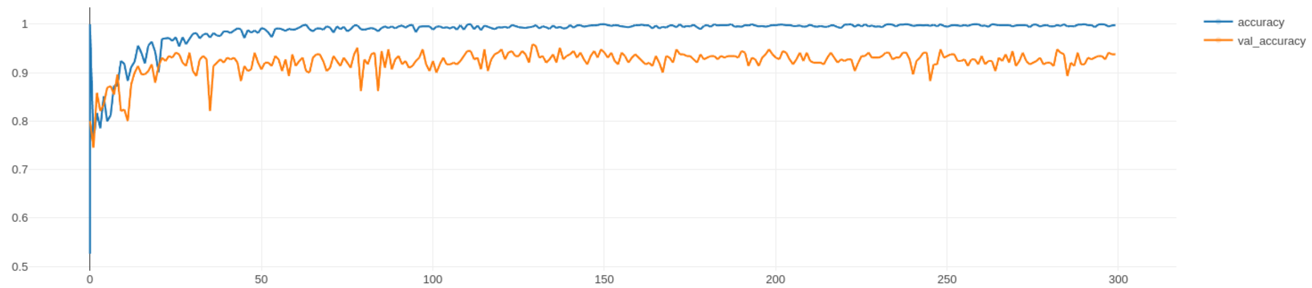
4 Painters

- 3 painters gave an nice result so the painter that was scraped was added
- This also gave an nice result but didn't make the 95% accuracy on the test dataset

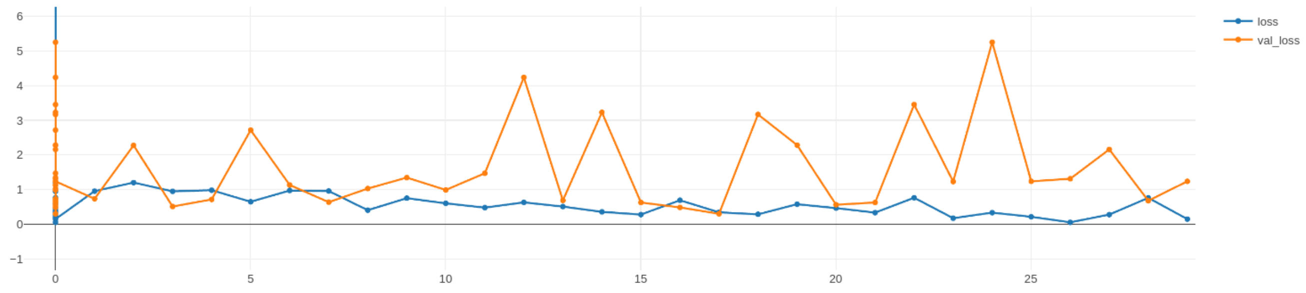


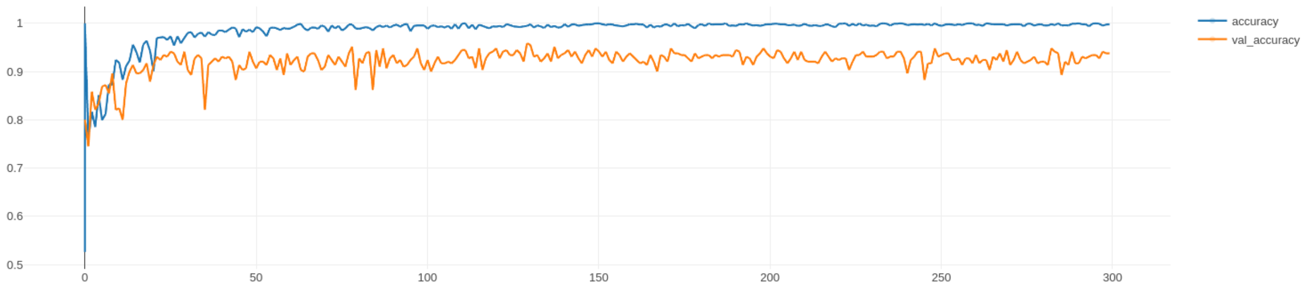
- Then a run with 300 epochs was started
  - This because even at 100 epochs there didn't seemed to be overfitting
  - When looking at this afterwards I looked at accuracy but should haved looked at the loss to detect overfitting





- Because even at 300 epochs the validation accuracy didn't realy drop I looked it up
- Now I found out I should have looked at loss to see when there is overfitting
- And restarted the training with 30 epochs





- the model didn't change for these final steps
- The test accuracy is just below 95%, but because 95% was already reached with 3 painters
    - there were no more improvements

```
data_augmentation = keras.Sequential([
            layers.RandomFlip("horizontal"),
            layers.RandomRotation(0.1),
            layers.RandomZoom(0.2),
            ])

        conv_base = keras.applications.vgg19.VGG19(
                    weights="imagenet",
                    include_top=False
                    )

        conv_base.trainable = True
        for layer in conv_base.layers[:-2]:
            layer.trainable = False


        inputs = keras.Input(shape=input_shape)
        x = inputs
        x = data_augmentation(x)

        x = keras.applications.vgg19.preprocess_input(x)
        x = conv_base(x)

        x = layers.Flatten()(x)
        x = layers.Dense(256, activation="relu")(x)
        x = layers.Dropout(0.5)(x)
        outputs = layers.Dense(output_shape, activation="softmax")(x)
        model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.compile(loss="categorical_crossentropy",optimizer="rmsprop",metrics=
["accuracy"])
```

Video website

See github for video (Or toledo)