

Logboek: CNN-model voor fractuurdetectie

Projectlogboek: Fractuurdetectie in de onderarm met Deep Learning

Doel van het project

Het doel is om een deep learning-model te trainen dat automatisch botbreuken in röntgenfoto's van de onderarm detecteert. Er wordt een gebalanceerde dataset samengesteld, gevolgd door preprocessing, filtering, augmentatie en training van een neuraal netwerk.

Dag 2 — Datasetselectie en preprocessing

Activiteiten

- Dataset bevatte röntgenfoto's van verschillende lichaamsdelen.
- Enkel afbeeldingen van de onderarm geselecteerd.
- Lege labels beschouwd als 'gezond'.
- De dataset is gebalanceerd naar:
 - 262 gebroken onderarmen
 - 262 gezonde onderarmen
- Alle afbeeldingen gecroppt en geschaald naar 64x64 pixels.
- Strategie uitgewerkt voor datasetversterking:
 - Contrastverhoging
 - Augmentatie door rotaties van bestaande afbeeldingen

Dag 3 — Filtering en contrastverbetering

Activiteiten

- CLAHE toegepast voor contrastverhoging.
- Sobel-filters toegepast voor randdetectie (X- en Y-richting).
- Gabor-filters toegepast met oriëntaties van 0°, 45°, 90° en 135°.
- Output van Gabor-filters samengevoegd in een 4-kanaals representatie.

Verantwoording

Gabor-filters zijn gekozen vanwege hun vermogen om directionele structuren zoals bottextuur te detecteren. Zie referentie: <https://www.semanticscholar.org/paper/On-estimating-the-directionality-distribution-in-Gdyczynski-Manbachi/213bc1b449edb93abd6fddc4ad9c4151be921c25>

Dag 4 — Filtering op donkere beelden en CLAHE-analyse

Activiteiten

- Op advies van de begeleider zijn alleen gezonde beelden geselecteerd die minder dan 10% zwarte pixels bevatten.
 - CLAHE verder geanalyseerd:
 - Verdeelt de afbeelding in 8×8 tegels.
 - Verschillende cliplimieten getest (30, 40, 50).
 - Histogram clipping toegepast met herverdeling van overtollige bins.
-

Dag 6 — Toevoegen validatieset

Activiteiten

- Validatieset toegevoegd (15% van de dataset), bestaande uit ongeveer 75 afbeeldingen.
 - Doel: het controleren van overfitting en het verbeteren van modelgeneraliseerbaarheid.
-

Dag 7 — Eerste modelstructuur

Modelstructuur

Conv2D(32) -> MaxPool -> Conv2D(64) -> MaxPool -> Conv2D(128) -> MaxPool -> Flatten -> Dense(128) -> Dropout -> Dense(2)

Resultaten

- Totaal aantal parameters: ~1.1 miljoen
- Training over 50 epochs leverde:
 - Train loss: 0.5098
 - Train accuracy: 74.91%
 - Validation loss: 0.5793
 - Validation accuracy: 70.51%

Observatie

Tekenen van overfitting waargenomen. Verdere augmentatie of architectuuraanpassing aanbevolen.

Dag 8 — Testsetvoorbereiding

Activiteiten

- Labels ingelezen en gesorteerd op categorie.
 - Enkel categorie 2 (fractuur in onderarm) geselecteerd.
 - Testset samengesteld met 50/50 verhouding tussen gezond en gebroken.
 - Enkel de eerste 9 elementen uit labels gebruikt (categorie + coördinaten).
-

Dag 9 — Testloop geïmplementeerd

Activiteiten

- Elke testafbeelding (512x512) in kleine blokken van 64x64 gepresenteerd aan het model.
 - Elk blok afzonderlijk geklassificeerd.
-

Dag 10 — Eerste testresultaten

Resultaten

- Model getraind met `binary_crossentropy`: `model.evaluate: loss = 0.8002, accuracy = 0.5000`

yaml Copy Edit

- Activatiefuncties gewijzigd:
- ReLU vervangen door Sigmoid in de hidden layers.
- Nieuwe evaluatie:

```
model.evaluate: loss = 0.7080, accuracy = 0.5000
```

Observatie

Verandering in activatiefunctie leidde niet tot verbetering.

Dag 11 — Data-augmentatie

Activiteiten

- Enkel de gebroken onderarmbeelden geaugmenteerd:
- Elke afbeelding geroteerd met 90° tot 4 variaties.
- Datasetgroottes:
- Gebroken: 1335 afbeeldingen
- Gezond: 667 afbeeldingen
- Model getraind met:
- ReLU in hidden layers
- Sigmoid in output layer

Epoch 24: Train loss: 0.2492 Train accuracy: 90.61% Validation loss: 0.7275 Validation accuracy: 71.79%

Observatie

Hoewel de validatie-accuratesse hoog is, is er sprake van overfitting. Verdere regularisatie en databalancing zijn nodig.

Tussencoclusie

- Dataset is vergroot en geoptimaliseerd.

- Preprocessing met CLAHE en Gabor-filters draagt bij aan detectie van botstructuur.
- Model vertoont overfitting, ondanks hoge accuratesse op de trainingsset.
- Testresultaten blijven steken op 50% accuraatheid.
- Volgende stappen:
 - Regularisatie toepassen (dropout, L2)
 - Teststrategie verbeteren
 - Complexere augmentatie toepassen

Dag 12

Datum: 18/06

Modelstructuur (eerste versie)

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape,
padding='same'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation= 'relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(num_classes, (1, 1), activation='sigmoid', name='output_layer')
])
```

Modelstructuur (tweede versie)

```
model = models.Sequential([
    layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(20, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(20, (5,5), activation= 'relu' ),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(10, (3,3), activation= 'relu' ),
    layers.Conv2D(num_classes, (3, 3), activation='sigmoid', name='output_layer'),
    layers.Flatten()
])
```

Resultaten

- Loss stijgt vanaf epoch 27. Overfitting begint daar.

- Gebruikte 27 epochs.
 - Validatie accuracy bij epoch 27: ~70%.
 - Te kleine testset -> uitgebreid met 20 fracture + 20 non-fracture images.
 - Resultaat: accurater, maar nog steeds ongebalanceerde dataset.
 - Oplossing: dataset balanceren door images met minder donkere pixels te selecteren.
-

Dag 13

Datum: 19/06

- Test-subset was niet genormaliseerd → nu wel gedaan.
- Nieuwe dataset:
 - 279 fracture images in totaal
 - 50 fracture / 50 non-fracture voor test
 - 20 fracture / 20 non-fracture voor train
- Veel images zijn al geaugmenteerd, dus niet opnieuw CLAHE toepassen
- Overfitting versnelt bij dubbele contrastversterking

Nieuwe strategie:

- Aantal kernels verminderen
- Andere activatiefuncties overwegen
- Augmentaties: rotatie, elastic curves, Gaussian noise
- Gaussian noise helpt om patronen in minder gunstige omstandigheden te detecteren

Code-issue opgelost:

```
# image_path was niet opnieuw gedefinieerd binnen tweede for-loop → opgelost.
```

Dag 14

Datum: 20/06

Test: minder kernels

```
def create_cnn_model(input_shape=(64,64,1), num_classes=1):
    model = models.Sequential([
        layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(10, (3,3), activation='relu'),
        layers.MaxPooling2D(2,2),
```

```

    layers.Conv2D(10, (5,5), activation= 'relu' ),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(10, (3,3), activation= 'relu' ),
    layers.Conv2D(num_classes, (3, 3), activation='sigmoid',
name='output_layer'),
    layers.Flatten()
])
return model

```

Resultaten:

- 27 epochs → stabielere leerproces, nog steeds lage accuraatheid
- 50 epochs → overfitting vanaf ~35 epochs
- Beste model op 30 epochs

Vergelijking:

- Confusion matrix 35 epochs: accuracy: 0.71
- Confusion matrix 30 epochs: accuracy: 0.75
- Confusion matrix 50 epochs: accuracy: 0.68

Gekozen model (30 epochs):

```

model = models.Sequential([
    layers.Conv2D(64, (5, 5), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(256, (5,5), activation= 'relu' ),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(256, (3,3), activation= 'relu' ),
    layers.Conv2D(num_classes, (3, 3), activation='sigmoid', name='output_layer'),
    layers.Flatten()
])

```

- Veel parameters → overfitting
- Maar goede performance bij testen

Dag 15

Datum: 21/06

Verbetering preprocess:

- Gaussian noise σ aangepast:
 - $\sigma = 0 \rightarrow$ te weinig
 - $\sigma = 0.5 \rightarrow$ te veel ruis
 - $\sigma = 1 \rightarrow$ goed compromis

Effect:

- Botstructuren zichtbaarder
- Textuurverschil tussen spier- en botweefsel duidelijker
- Leidt mogelijk tot betere fractuurdetectie

Volgende stap:

- Lichter model trainen met minder parameters
 - Nog niet volledig afgelond
-

To do:

- Meer data verzamelen zonder herhaling
- Dataset verder balanceren
- Augmentatie optimaliseren (geen overmatige contrastaanpassing)
- Optimalisatie activatiefuncties en regulatie (Dropout?)
- Mogelijk overstappen naar transfer learning (bij kleine datasets)

CNN Model Training Logboek

Day 16

Ik merkte dat de meeste afbeeldingen in de trainingsset al aangepast waren met een ander soort contrast, wat betekent dat veel afbeeldingen al drie of vier keer herhaald werden.

Als ik **CLAHE** meerdere keren toepas, zal het model **sneller overfitten**. Daarom ga ik proberen het aantal **kernels** en het type **activatiefuncties** te veranderen.

Wijzigingen in augmentatie:

- Rotatie was al toegepast.
 - Nieuwe augmentaties:
 - **Elastic Curve**
 - **Gaussian Noise**
- Dit maakt het mogelijk dat het model patronen van fractures herkent onder slechtere omstandigheden.

Foutopsporing:

- Er was een **probleem met variable scoping** in de **store_label** functie.
- **image_path** werd in de tweede **for**-lus gebruikt zonder daar opnieuw te zijn gedefinieerd; Python gebruikte daardoor de variabele uit de eerste **for**-lus.

Day 17 — 20/06

Vandaag heb ik getest of **wijziging van het aantal kernels** invloed heeft op overfitting.

Ik merkte dat een groter aantal kernels meer features oplevert, maar ook tot meer **overfitting** leidt.

Modelconfiguratie:

```
def create_cnn_model(input_shape=(64,64,1), num_classes=1):
    model = models.Sequential([
        layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(10, (3,3), activation='relu'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(10, (5,5), activation= 'relu'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(10, (3,3), activation= 'relu'),
        layers.Conv2D(num_classes, (3, 3), activation='sigmoid',
        name='output_layer'),
        layers.Flatten()
    ])
    return model
```

Trainingsresultaten: 27 epochs met kernels [20, 10, 10, 10, 1]: stabielere leercurve, lage accuraatheid, geen sterke overfitting.

50 epochs: overfitting treedt op vanaf ongeveer 35 epochs.

35 epochs gaf de laagste verlieswaarde.

Vergelijking modellen: Model (Epochs) Mean Accuracy 35 0.71 30 0.75 (beste) 50 0.68

Ik heb het 30-epoch model gebruikt om te testen op grotere afbeeldingen door deze in 64x64 slices te verdelen.

Structuur van best presterende model:

```
model = models.Sequential([
    layers.Conv2D(64, (5, 5), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(256, (5,5), activation='relu'),
    layers.MaxPooling2D(2,2),
```

```

    layers.Conv2D(256, (3,3), activation='relu'),
    layers.Conv2D(num_classes, (3, 3), activation='sigmoid', name='output_layer'),
    layers.Flatten()
])

```

dag 18

Dagboek - Botfractuurdetectie met CNN

Day 18: 21/06

Vandaag heb ik de `xsigm`-parameter van het Gaussiaanse filter aangepast naar 1.0. Aanvankelijk stond deze op 0, wat te laag is. Daarna probeerde ik een waarde van 0.5, maar dat gaf te veel ruis. Bij een waarde van 1 was de ruis aanzienlijk minder, waardoor donkere niveaus tussen botstructuren beter zichtbaar werden.

Deze verbetering helpt het model om potentiële kraakbeenruimtes of lege ruimtes (mogelijk een breuk) te detecteren. De textuur van het bot wordt duidelijker zichtbaar met deze aanpak en onderscheidt zich veel beter van spierweefsel.

Modeloptimalisatie: eenvoudiger model met minder parameters

Daarna probeerde ik het model te verbeteren door over te stappen naar een kleiner model met minder parameters:

```

def create_cnn_model(input_shape=(64,64,1), num_classes=1):
    model = models.Sequential([
        layers.Conv2D(10, (5, 5), input_shape=input_shape, padding='same'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(20, (3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(20, (5,5), activation='relu', padding='same'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(10, (3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(num_classes, (3,3), activation='sigmoid', padding='same'),
        layers.MaxPooling2D(4,4),

        layers.Flatten()
    ])
    return model

```

Ik heb dit model getraind met 20 epochs. De binary crossentropy-curve zag er goed uit — de training- en validatieverlies waren dicht bij elkaar. Na de training verkreeg ik de volgende confusion matrix: "configuration

matrix 20 epochs less parameters 10 10 20 20.png"

Testen op grotere afbeeldingen Om grotere afbeeldingen te testen, splits ik de afbeelding op in 64x64 blokken en voorspel ik elk blok afzonderlijk. "some images are confusing the model.png"

Sommige testafbeeldingen zijn verwarring voor het model. Bijvoorbeeld: patronen in het gips doen het model denken dat er een breuk is. De aanwezigheid van gipshoezen vermindert de nauwkeurigheid van het model, omdat de textuur van het gips sterk lijkt op die van een botbreuk op röntgenbeelden. Hierdoor kunnen AI-modellen moeite hebben met het onderscheiden van echte fracturen en gips, wat leidt tot foutpositieven of verwarring bij de classificatie. zoals in deze afbeelding:

Nieuw model met betere prestaties (81% accuraatheid)

Na meerdere pogingen ontwikkelde ik het volgende model:

```
model = models.Sequential([
    layers.Conv2D(10, (5, 5), input_shape=input_shape, padding='same'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(25, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(25, (5,5), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(10, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(num_classes, (3,3), activation='sigmoid', padding='same'),
    layers.MaxPooling2D(4,4),

    layers.Flatten()
])
```

Deze versie haalde een accuraatheid van 81%, maar raakt nog steeds in de war bij gezonde botstructuren of texturen die op fracturen lijken.

Grotere afbeeldingen, nauwkeurigere voorspellingen

Bij gebruik van grotere afbeeldingen en het volgende model: def create_cnn_model(input_shape=(64,64,1), num_classes=1): model = models.Sequential([layers.Conv2D(10, (5, 5), input_shape=input_shape, padding='same'), layers.MaxPooling2D((2, 2)),

```
    layers.Conv2D(30, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(25, (5,5), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),
```

```

        layers.Conv2D(10, (3,3), activation='relu', padding='same'),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(num_classes, (3,3), activation='sigmoid', padding='same'),
        layers.MaxPooling2D(4,4),

    layers.Flatten()
])
return model

```

Na 50 epochs vond dit model fracturen op drie grote testafbeeldingen. Echter, op afbeelding 3 faalt het om onderscheid te maken tussen complexe texturen van de pols en een daadwerkelijke fractuur:

"PREDICTED model 50 epochs weinig parameters.png"

Confusion matrix:

"configuration matrix 50 epochs less kernels.png" Accuracy: 79.2%

100 Epochs — Betere detectie, slechtere validatie Bij een training met 100 epochs verminderde de training loss, maar de validatie binary crossentropy bleef hoog:

"100 epochs crossentropy loss lower n of params.png" Accuracy: 73.2%

Het model detecteert wel een fractuur in afbeelding 3, maar met minder zekerheid:

"100 epochs crossentropy less kernels id 4.png"

Day 19

Vandaag heb ik geprobeerd de accuraatheid te verbeteren door dense-lagen toe te voegen. Dense-lagen abstracteeren de ruimtelijke informatie en focussen op concepten zoals scherpte en textuur — nuttig om bot van gips te onderscheiden.

Maar dit verhoogt ook het risico op overfitting, omdat het aantal trainbare parameters sterk toeneemt. Zoals eerder gezien, leidt een groter model op een kleine dataset vaak tot lagere prestaties. Model met dense-lagen

```

def create_cnn_model(input_shape=(64,64,1), num_classes=1):
    model = models.Sequential([
        layers.Conv2D(10 , (5, 5), input_shape=input_shape, padding='same'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(20,(3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(20, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),

        layers.Flatten(),
    ])
    return model

```

```
        layers.Dense(10, activation='relu'),
        layers.Dense(5, activation='relu'),
        layers.Dense(num_classes, activation='sigmoid')
    ])
return model
```

dag 20: learning rate verander ik net voordat overfitting optreedt, wat rond 20 epochs is.

Dag 21:

om overfitting te verminderen, ga ik dropout layers te implementeren. Dit helpt om de generaliseerbaarheid van het model te bevorderen door willekeurig een deel van de neuronen uit te schakelen tijdens training en weer te activeren. Dit voorkomt dat het model te afhankelijk wordt van specifieke neuronen en dat het model te specifiek is op patterns uit de training set.

de model die getraind wordt is en wordt geen learning rate aangepast:

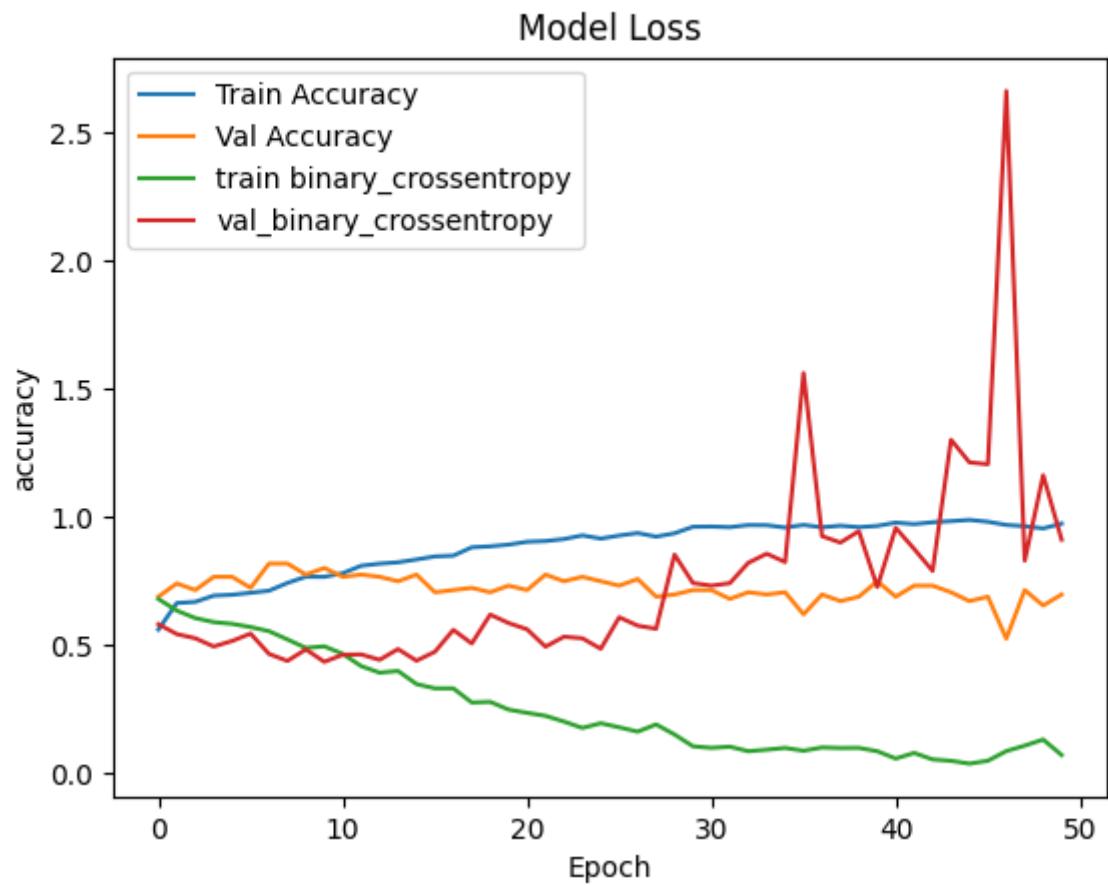
```
layers.Conv2D(25 , (5, 5), input_shape=input_shape),# 50
layers.MaxPooling2D((2, 2)),#output size :(30,30,20)
layers.Dropout(0.2),

layers.Conv2D(30,(3,3), activation='relu'),
layers.Dropout(0.2),

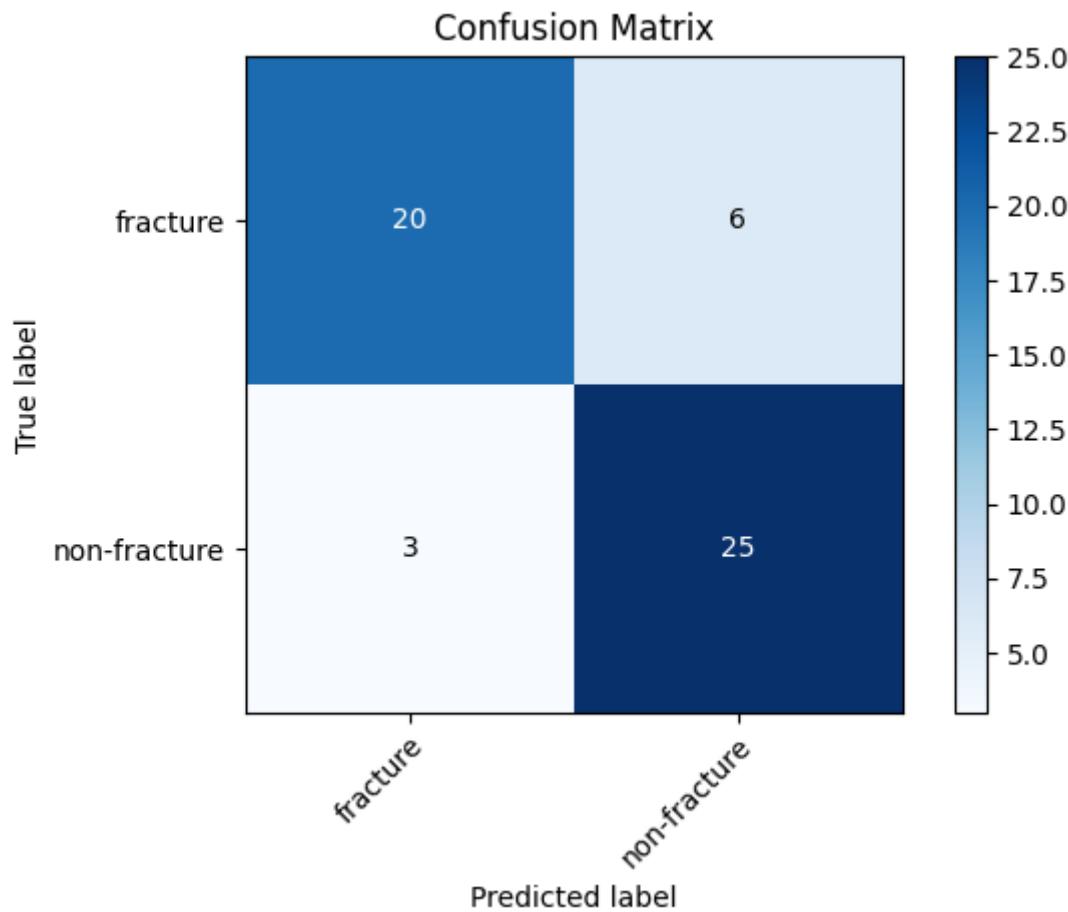
layers.MaxPooling2D((2,2), ),#output: (14,14,30),
layers.Conv2D(60,(3,3), activation='relu'),

layers.MaxPooling2D((2,2)),#output: (6,6,25)
layers.Conv2D(100, (3,3), activation='relu'),# 32 because it gives more
features for curves and edges
layers.Dropout(0.2),

layers.MaxPooling2D((2,2)),# output size :(2,2,20)
layers.Conv2D(num_classes, (2,2), activation='sigmoid'),
layers.Flatten(),
```

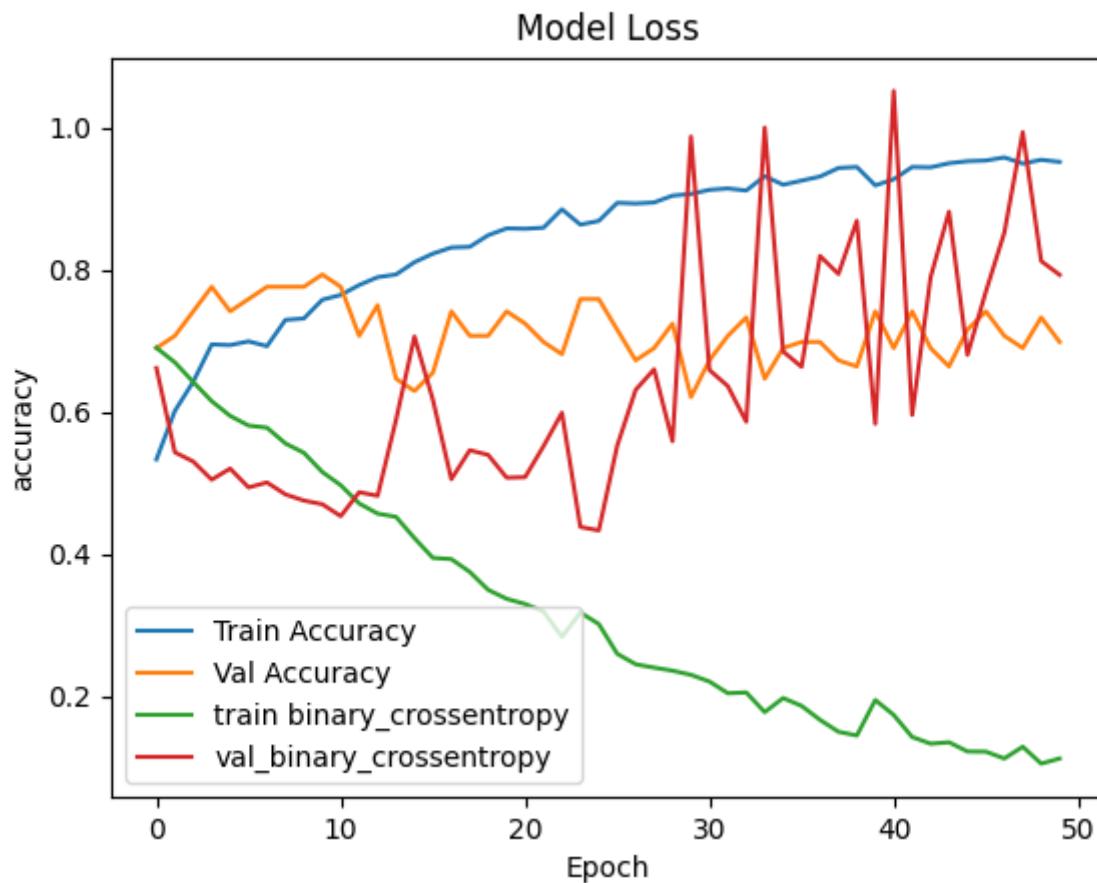


we zien dat de validatie loss (val crossentropy) nog steeds stijgt, wat betekent dat het model nog steeds overfit.
Ik ga proberen met een kleinere aantal kernels te trainen, maar met dropout layers

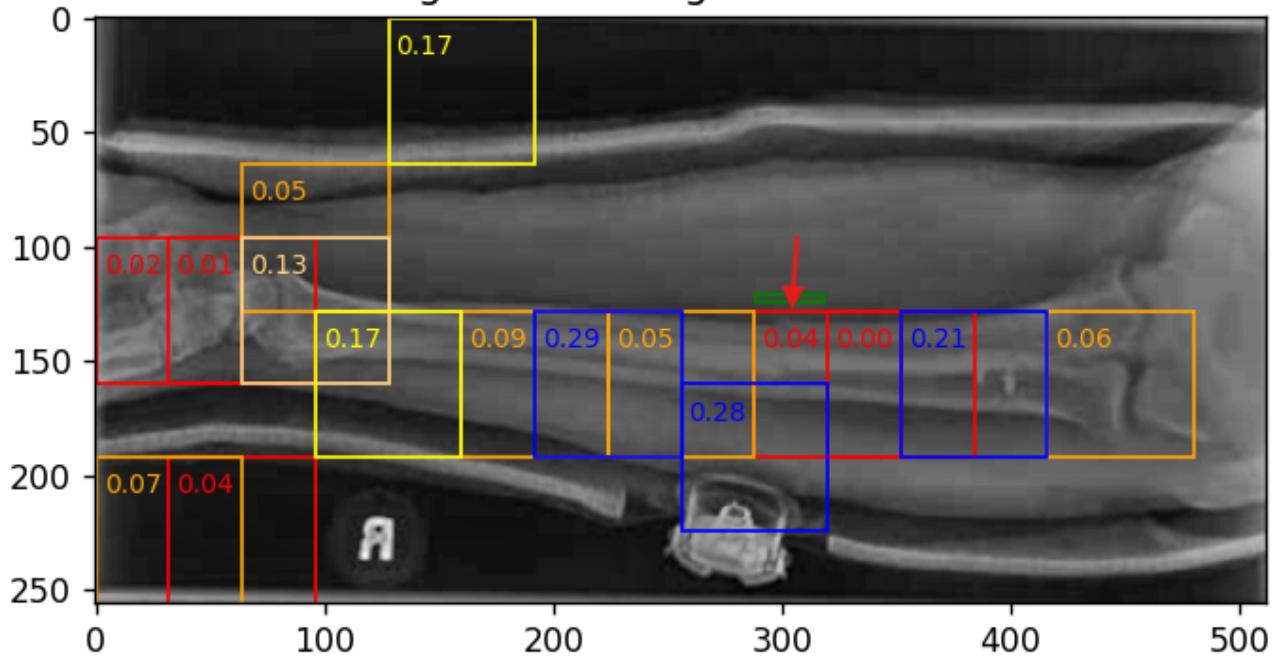
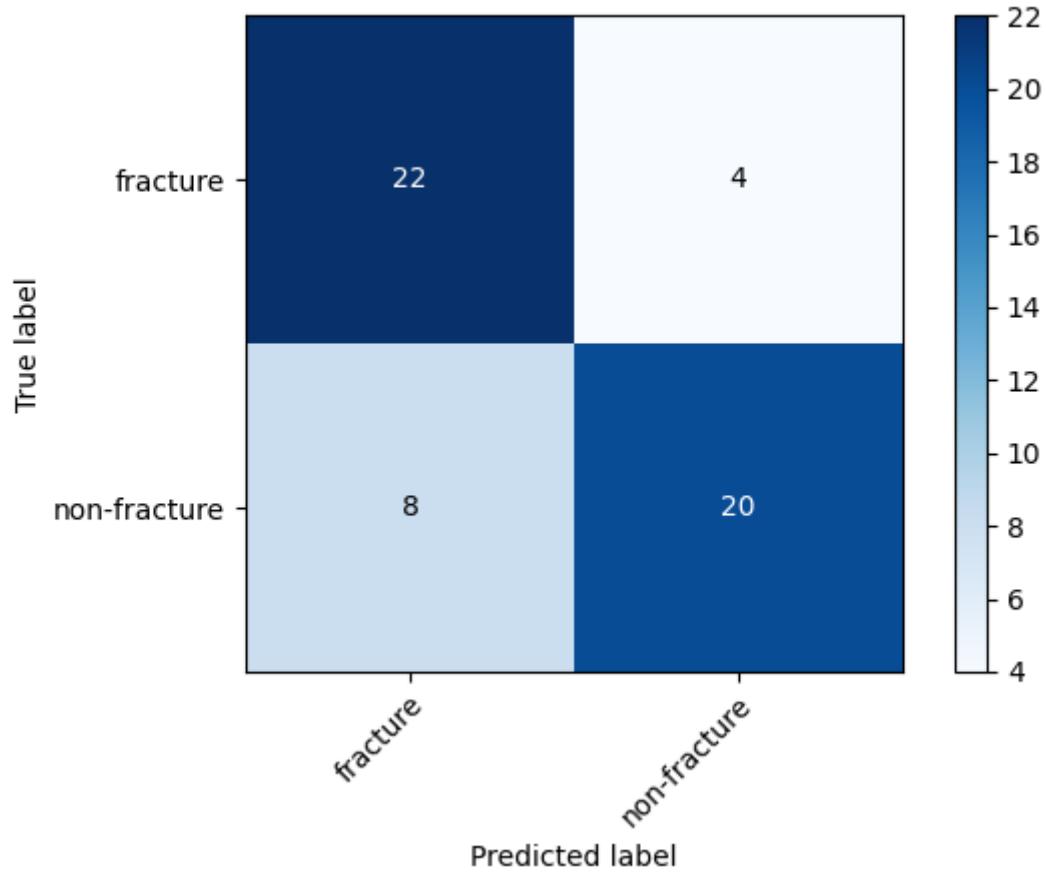


accuracy is 81% voor de classificatie van de fracturen. Dit is een verbetering ten opzichte van het model zonder dense-lagen, dat 77% accuraatheid behaalde.

vervolgens zijn er minder kernels gebruikt en het heeft de validatie loss verminderd na 30 epochs, maar overfitting blijft nog een probleem na 10 epochs. zie plaatje hieronder

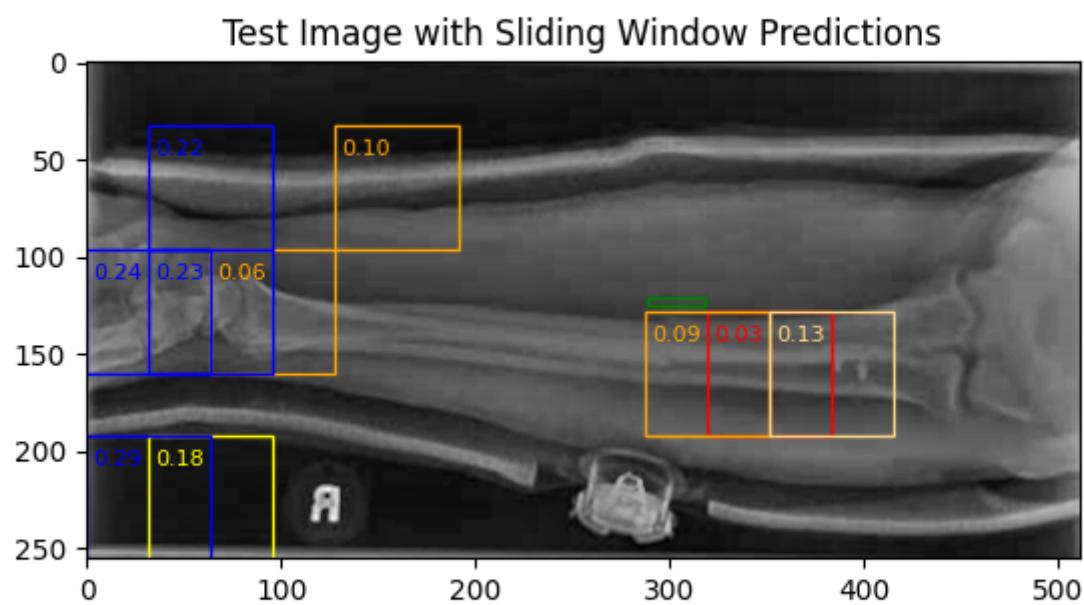
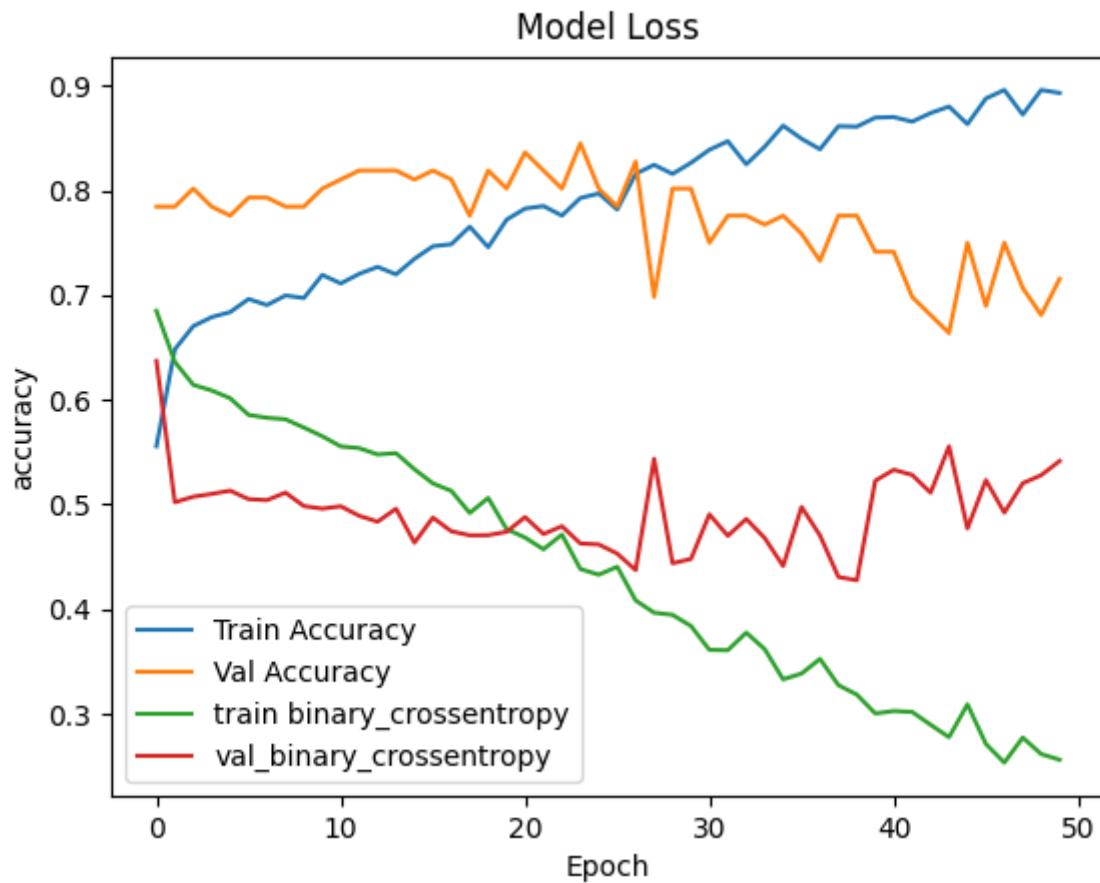


na het testen van dit model merk ik dat hij wel aangeeft dat de fractuur in zit, maar raakt afgeleid met randjes zoals giphoezen. De randjes van de vierkant zijn gekleurd op basis van hoe zeker het model is dat er een fractuur zit. De rode kleur betekent dat het model zeker is dat er een fractuur zit, terwijl blauw (koud) betekent dat het model het niet zeker weet. Met een de rode pijl wordt er aangegeven waar de true label in zit.

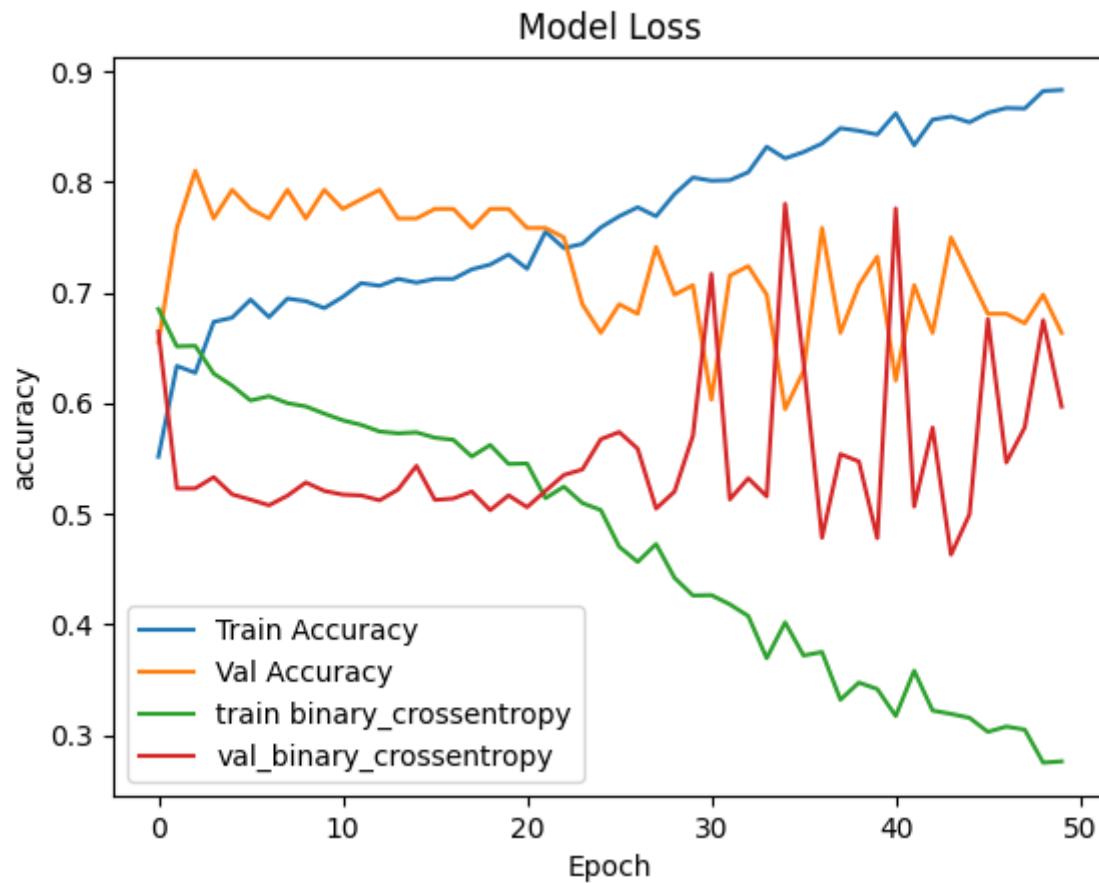
Test Image with Sliding Window Predictions**Confusion Matrix**

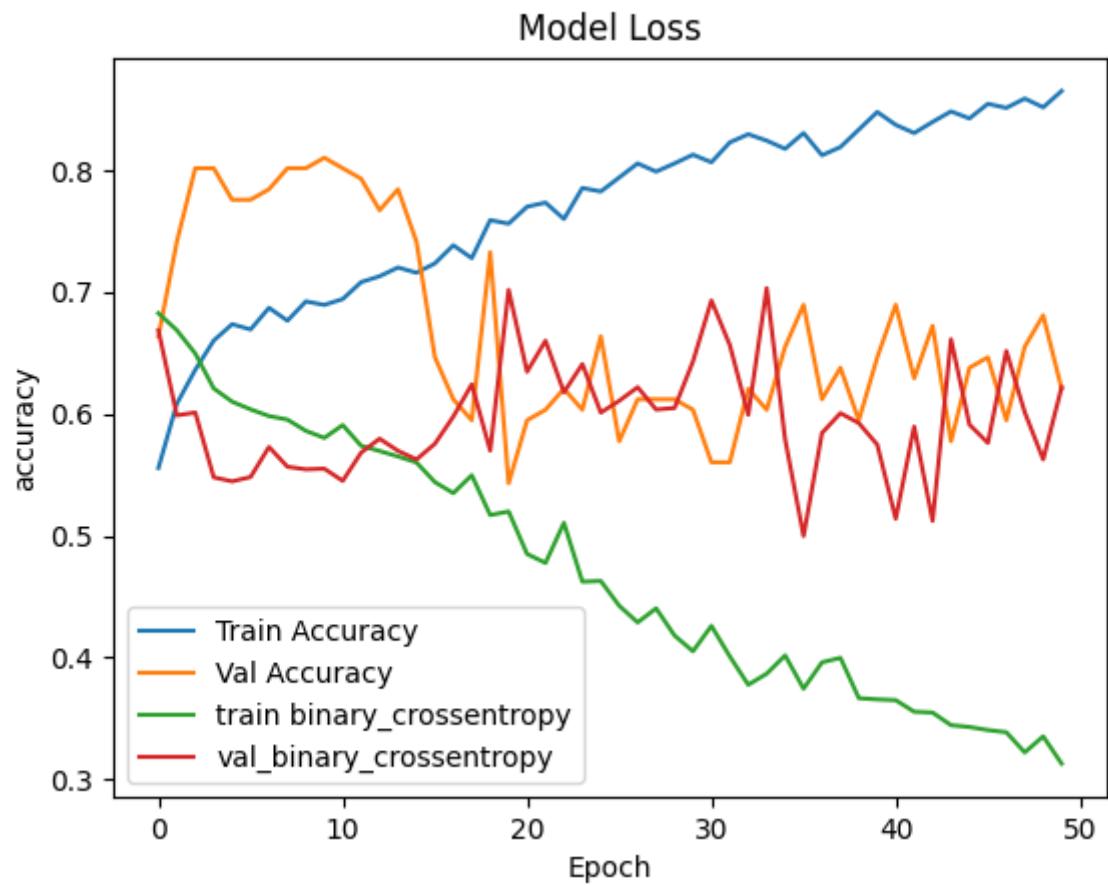
Accuracy: 77% Nu ga ik gaussian model proberen met dropout layers en dense layers.

Nu heb ik het aantal dropout layer verhoogd naar 0.2 en krijg deze loss curve:

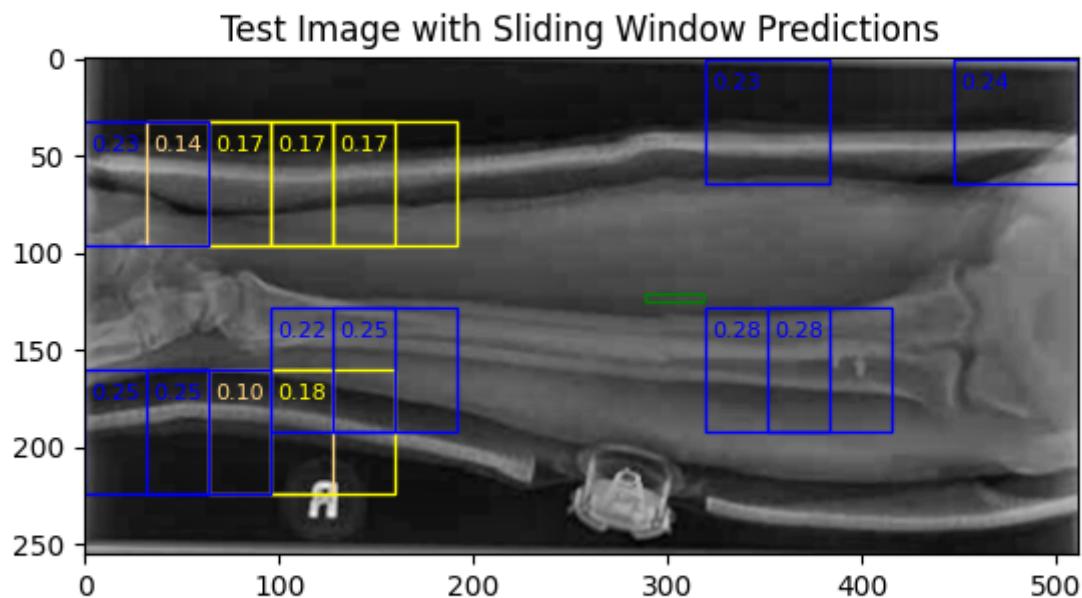


Nu heb ik het aantal dropout layer verhoogd naar 0.3 en krijg deze loss curve:



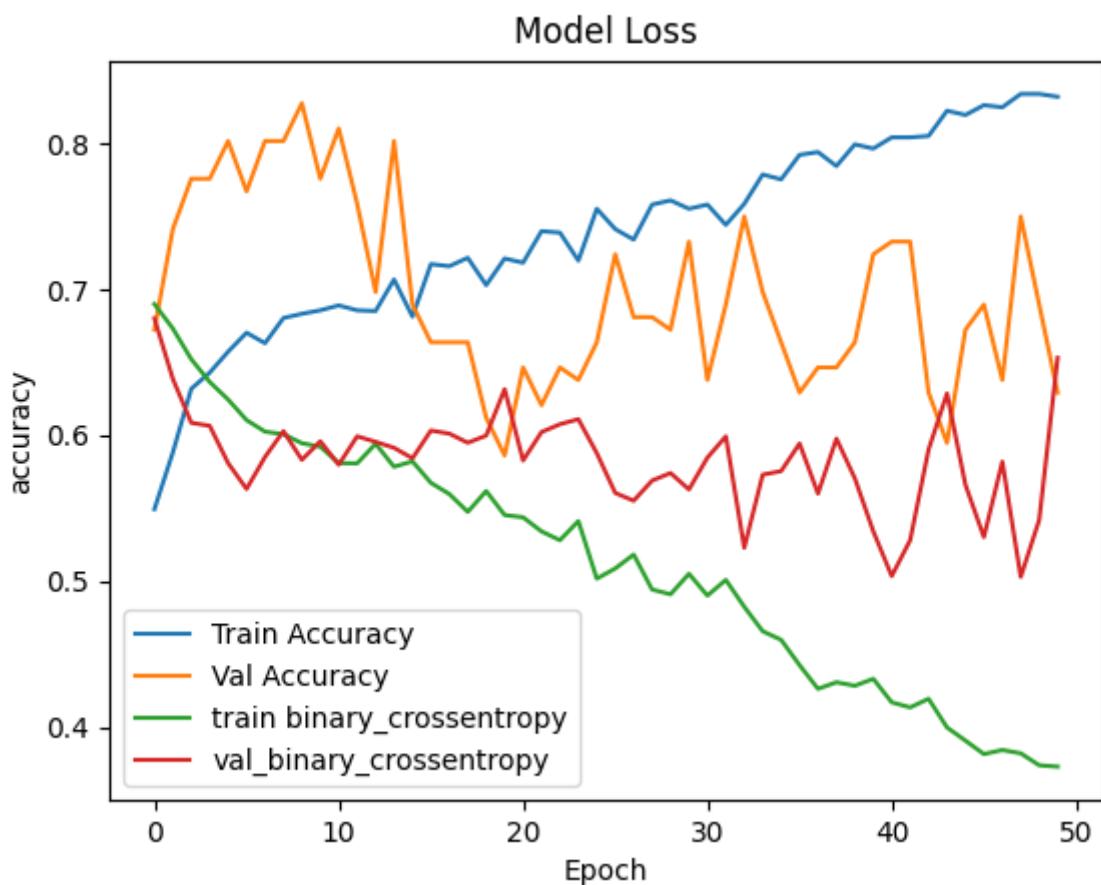


Deze loss curve geeft aan dat het model nieuwe data kan voorspellen maar dat hij minder zeker is van zijn uitkomst. Dit kunnen we zien in deze voorsellingen hieronder: in de afbeelding zijn er geen enkele voorsellingen met een 0.1 of lager wat betekent dat het model wel minder zeker is van zijn voorspelling. dit is een wijze van generalisatie. het model is minder "zeker" op nieuwe data maar kan wel voorspellen.

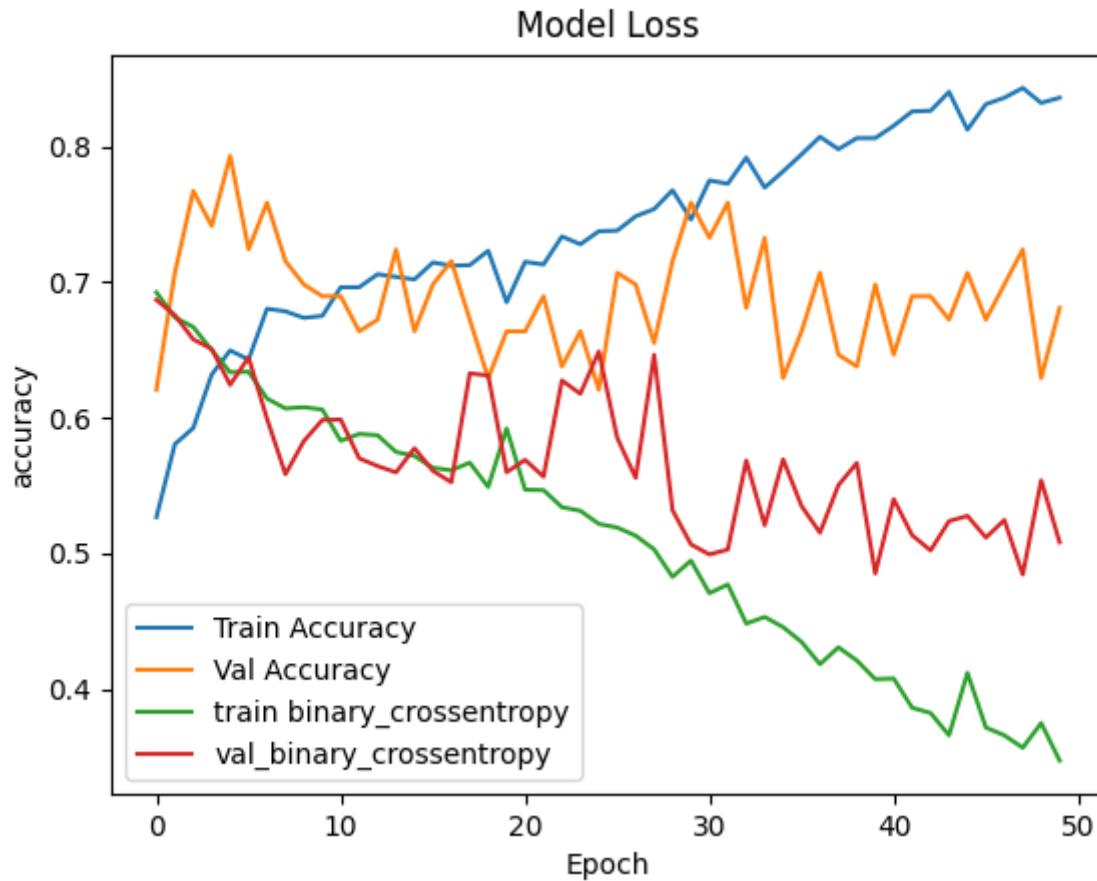


De

accuracy blijft 77% nu heb ik 0.5 dropout geprobeerd en krijg de volgende loss curve:

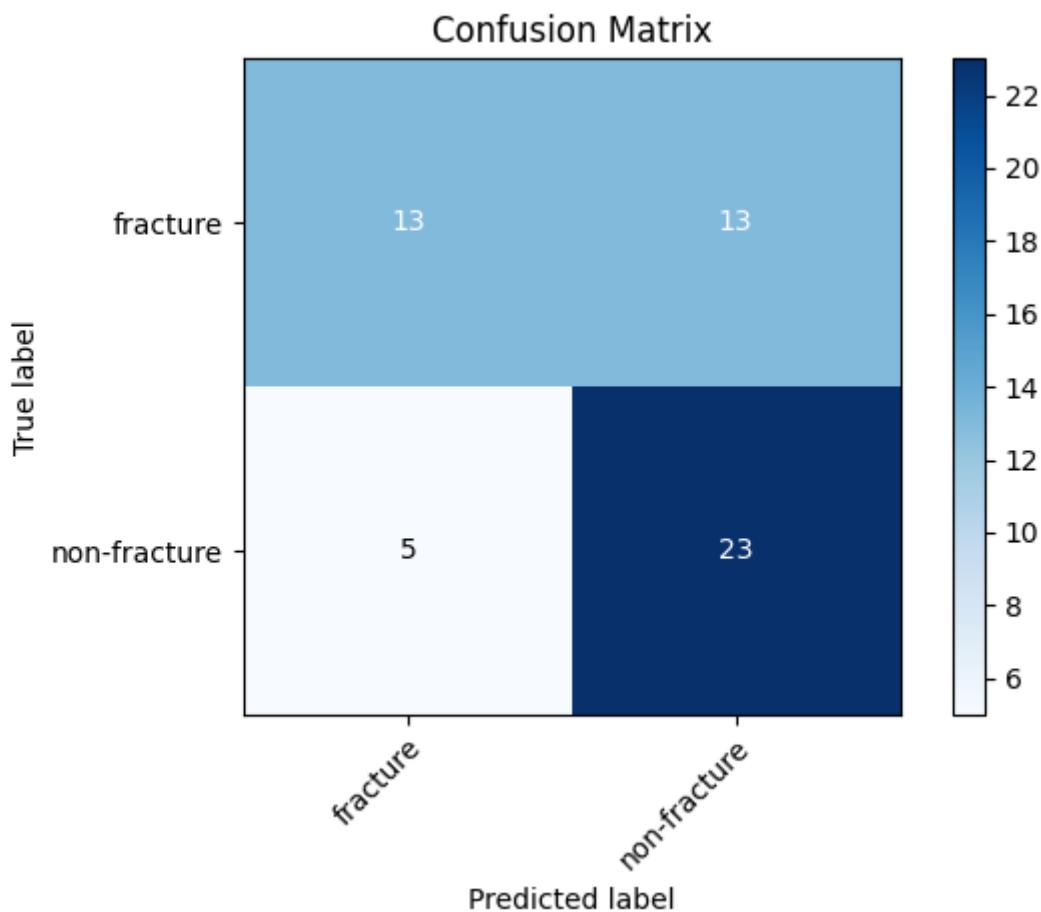


Nu heb ik het model getraind met twee gaussian layer en krijg ik de volgende loss curve:

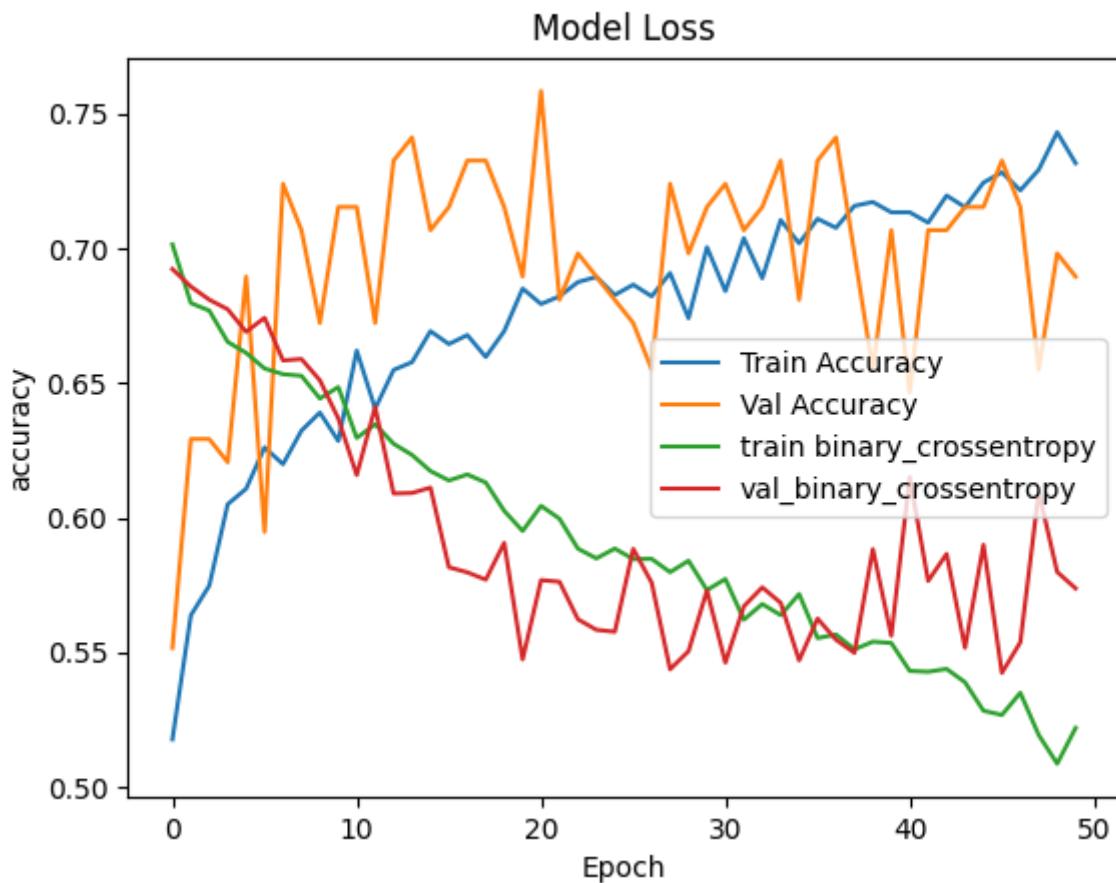


gaussian noise is een manier om het model te trainen met ruis. Zodat het model kan generaliseren en niet te veel overfit. We zien dat de val loss curve minder snel stijdt wat een voordeel is van heet model. Nu ga ik het

model testen met de sliding window methode.



Accuracy is 66% we zien dat door generalisatie de accuracy is gedaald en nu denkt het model dat meest van de fracturen geen fracturen zijn. Nu ga ik het model trainen met een gaussian noise van 0.2 en dense layers.



Daarna heb ik het model aangepast naar meer kernels omdat er nu meer variatie wordt opgeleverd met behulp van ruis.

```

layers.Conv2D(25, (5, 5), input_shape=input_shape),# 50 because it gives
more features for curves and
layers.MaxPooling2D((2, 2)),#output size :(30,30,20)
layers.GaussianNoise(0.2),

layers.Conv2D(50,(3,3), activation='relu'),
layers.Dropout(0.2),
layers.MaxPooling2D((2,2), ),

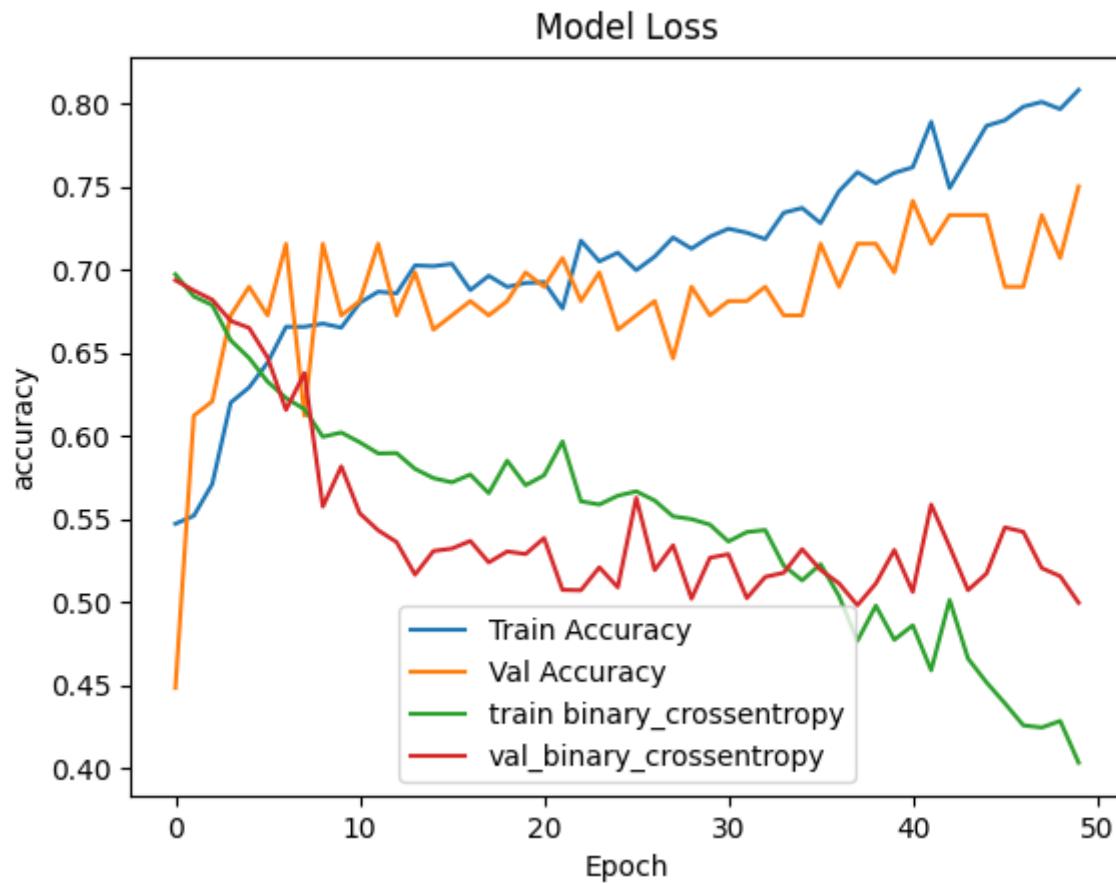
layers.GaussianNoise(0.2),
layers.Conv2D(50,(3,3), activation='relu'),
layers.Dropout(0.2),
layers.MaxPooling2D((2,2), 

layers.GaussianNoise(0.2),
layers.Conv2D(50, (3,3), activation='relu'),# 32 because it gives
layers.Dropout(0.2),

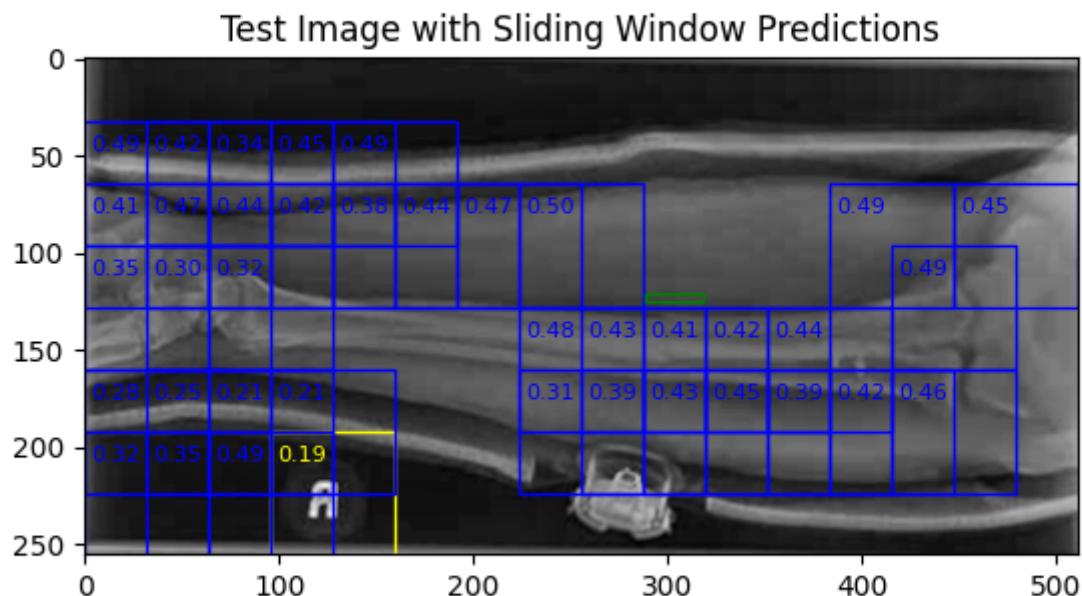
layers.MaxPooling2D((2,2)),# output size :(2,2,20)
layers.Conv2D(num_classes, (2,2), activation='sigmoid'),# 32 because
layers.Flatten(),

```

hiervan kreeg ik de volgende loss curve:



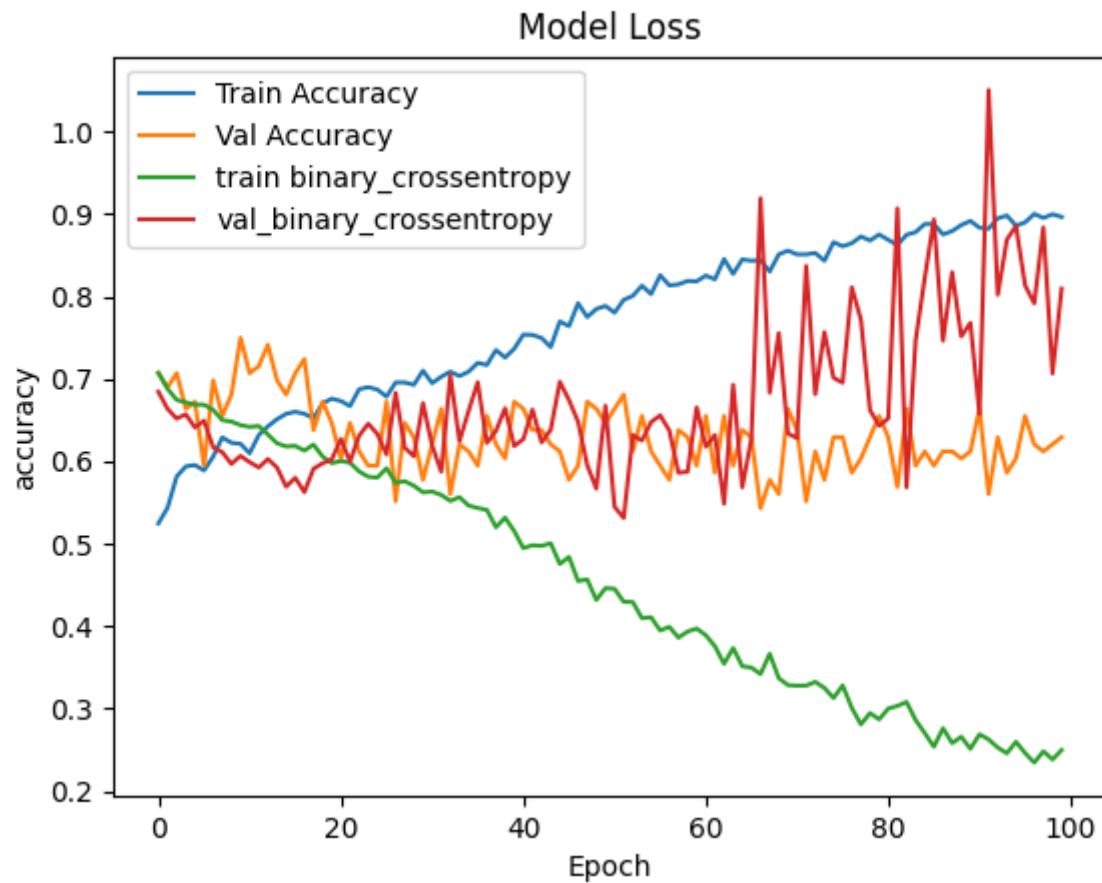
getest met sliding window methode

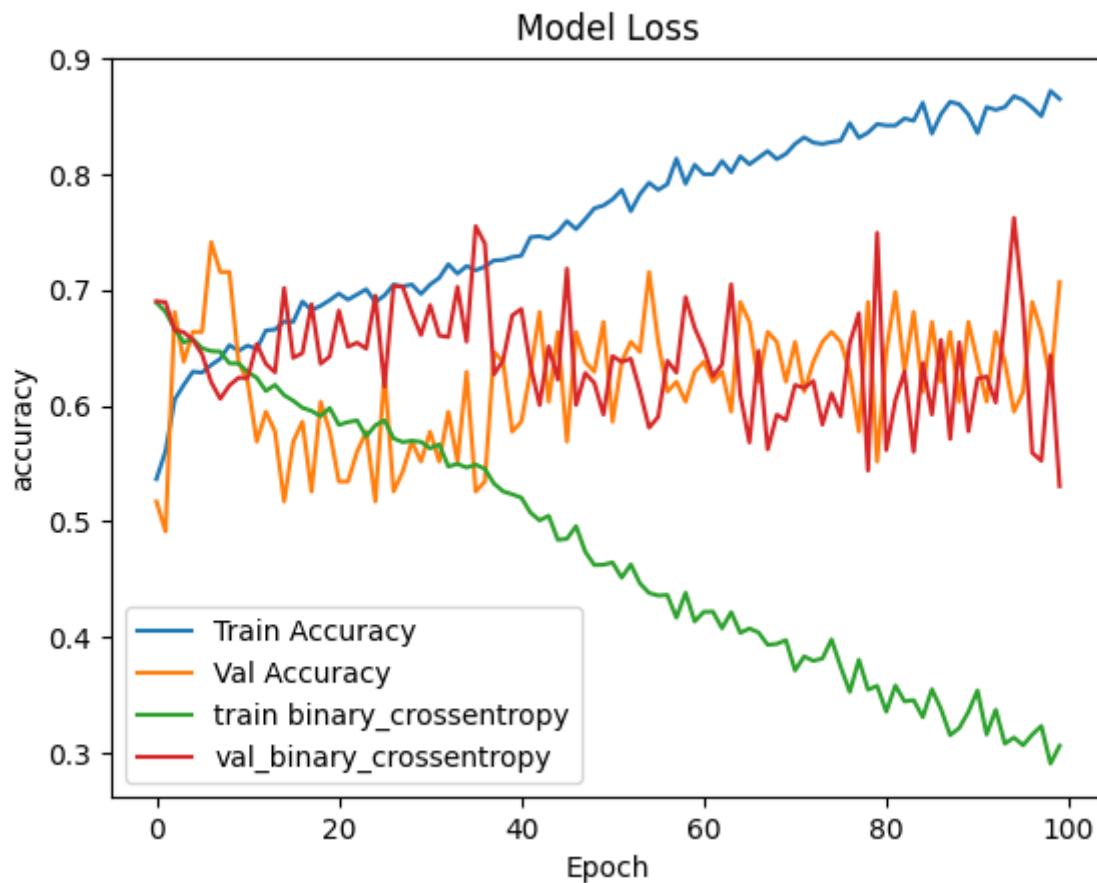


Dit betekent dat het model beter kan onderscheiden tussen botten en de rest de accuracy is hoger 80% en de val loss is ook lager. Ik ga meer epochs, dropouts en kernels toevoegen naarmate de loss curve beter wordt

na het bekijken van de dataset ben ik erachter gekomen dat sommige plaatjese hel donker zijn en dat het model daarme in de waar raakt. Het komt doordat iik de 64 px moet croppen en blykbaar sommige labels zijn

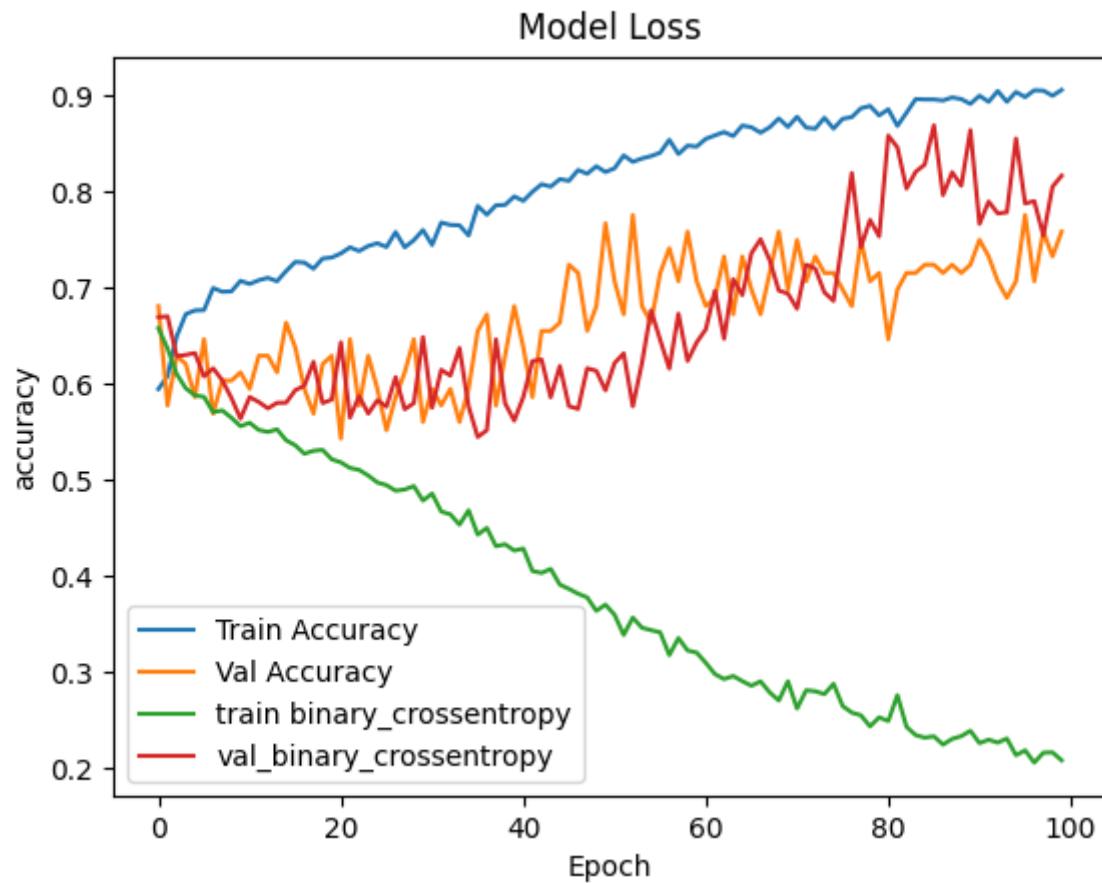
hniet heel duidelijk



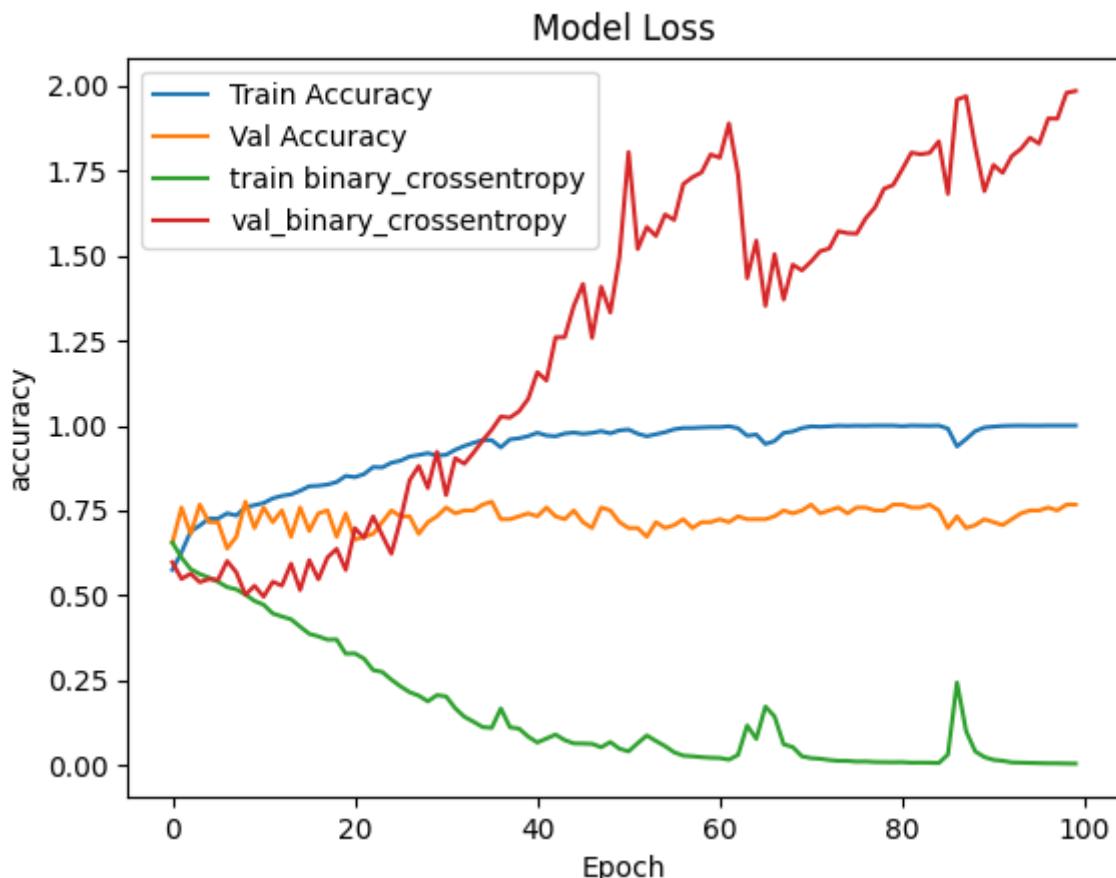


omdat generalisatie ervoorzorgt dat het model meer naar basische en algemene features kijkt, lijken basische edges nog steeds maar het lijkt lasof er een uncertainty zone tussen 0,3 en 0,75 die smaller wordt naarmate de generalisatie afneemt en het model meer zeker is van zijn voorspelling. ik heb besloten het aantal dropout kernels te verlagen naar 0.1 en de gaussian noise naar 0.2 om te zien of de generalisatie beter wordt aangepaakt door het model

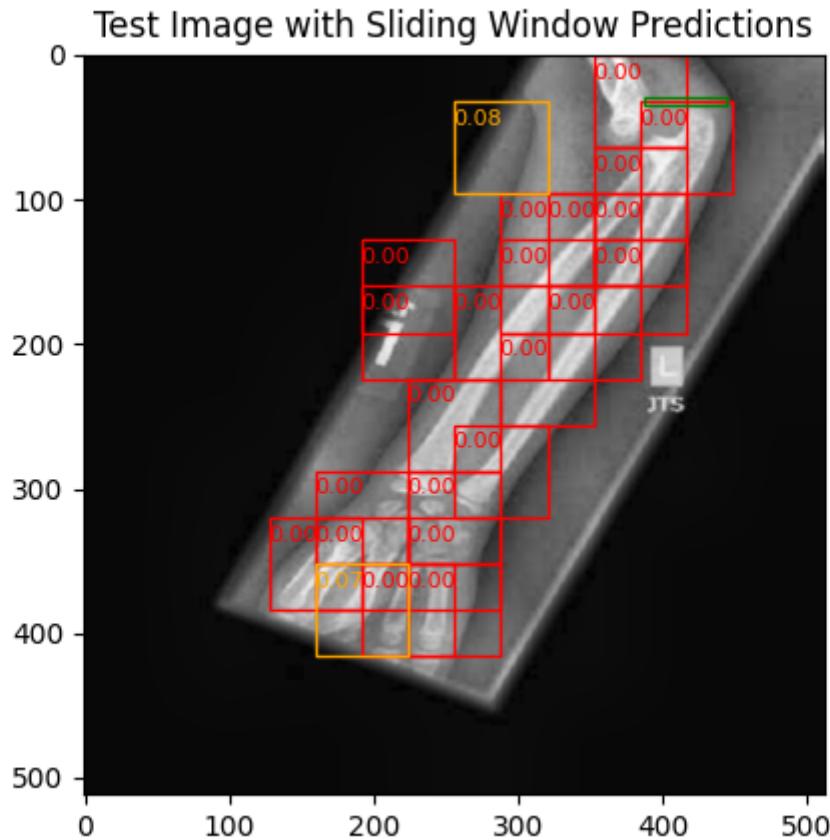
als het resultant niet afwijkt, dan ga ik de gaussian noise verlagen naar 0.1



Accuracy van 75% maar kan beter verschil maken tussen wat een bot em het randje van de image

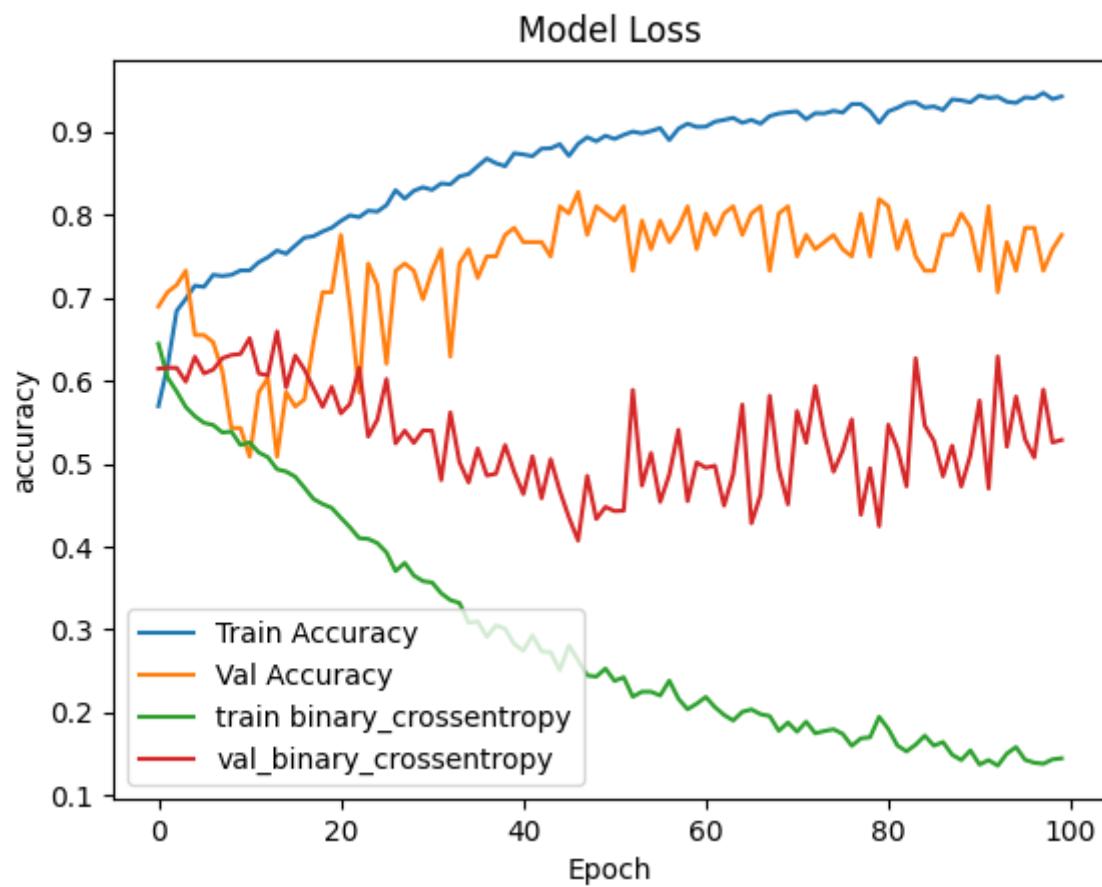


bij het testen zie ik dat het model zeer zeker is dat een bot een botbreuk is, zoals te zien is in dit plaatje:



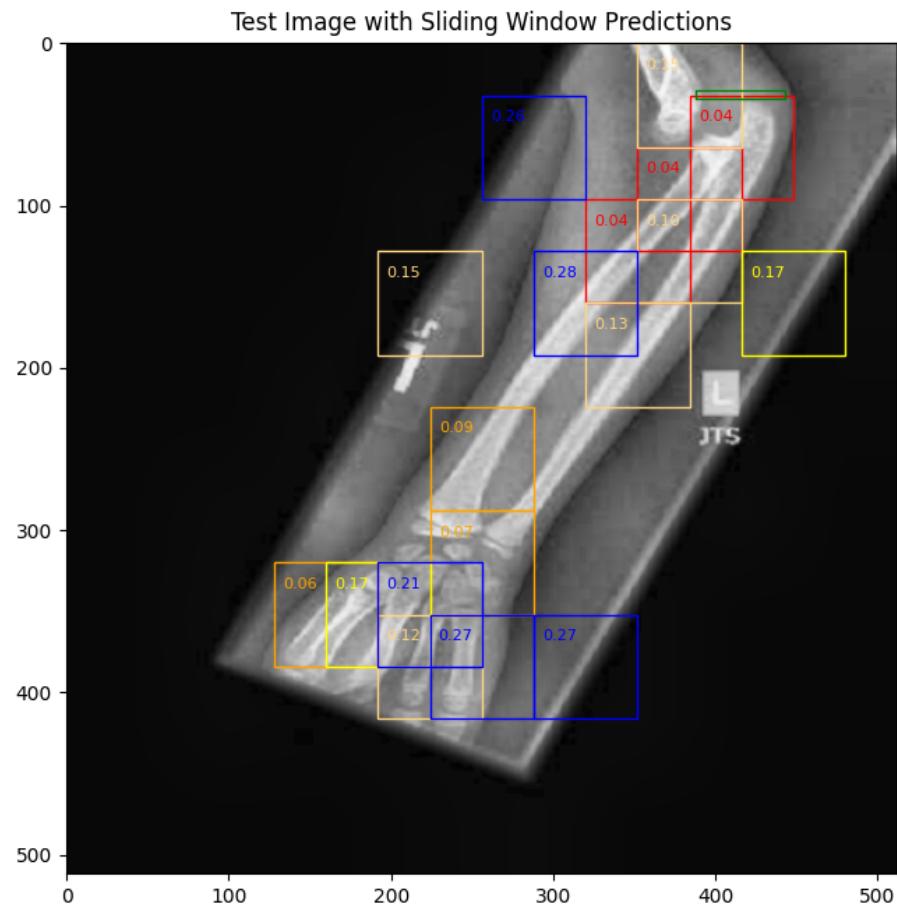
dag 22:

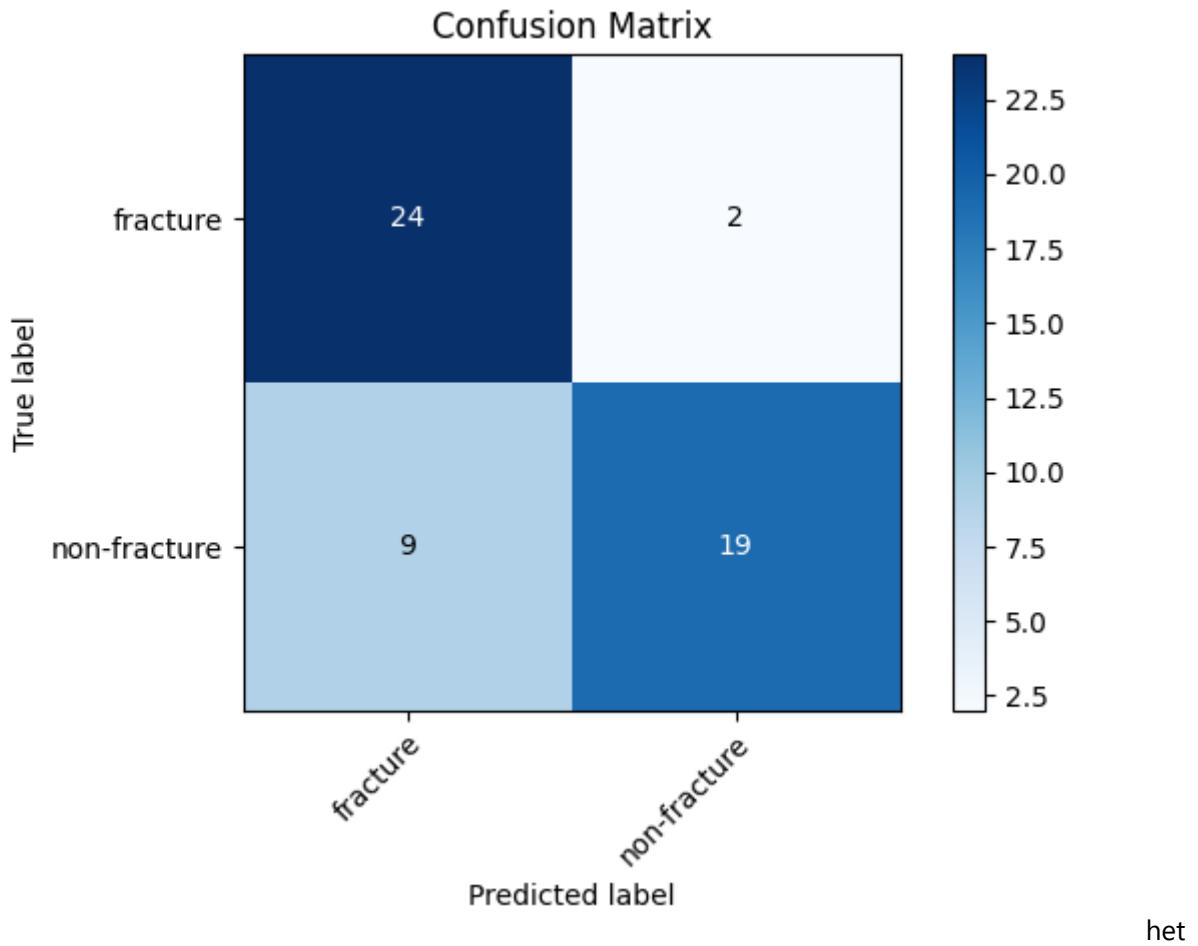
na het bekijken van de fotos ben ik erachter gekomen dat de originele foto's niet goed waren gecropt. In de gecropte afbeeldingen zaten delen van randjes van de afbeelding wat uiteraard niet de geweste situatie is. Het kwam erop neer dat ik de resizing van de afbeelding vergat. Nu dat ik dit heb gefixt is de loss curve beter geworden. Met dropout op 0.3 krijg ik de volgende loss curve:



de detectie test met sliding window methode

dit is





model heet: model = load_model('my_model_25_epochs_25')

dag 23

ik heb vervolgens het aantal kernels weer hoger gesteld met dit model

```
# layers.Conv2D(15 , (5, 5), input_shape=input_shape, padding='same'),# 50
because it gives more features for curves and
# layers.GaussianNoise(0.1, input_shape=input_shape),
layers.Conv2D(32 , (5, 5), input_shape=input_shape,activation = 'relu'),#
50 because it gives more features for curves and
layers.Dropout(0.3),
layers.MaxPooling2D((2, 2)),#output size :(30,30,20)

# layers.GaussianNoise(0.1),
layers.Conv2D(32,(3,3), activation='relu'),
layers.Dropout(0.3),
layers.MaxPooling2D((2,2), ),#output: (14,14,30),

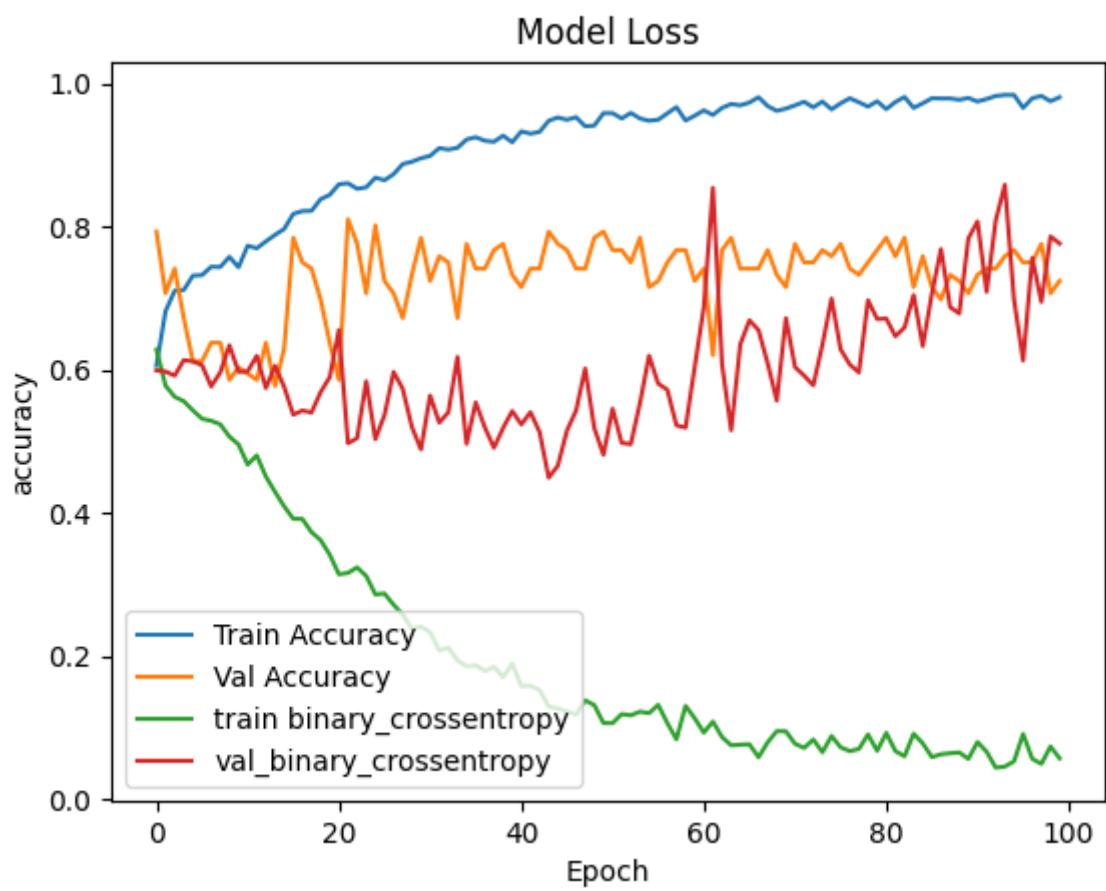
# layers.GaussianNoise(0.1),
layers.Conv2D(50,(3,3), activation='relu'),
layers.Dropout(0.3),
layers.MaxPooling2D((2,2)),#output: (6,6,25)

# layers.GaussianNoise(0.1),
layers.Conv2D(100, (3,3), activation='relu'),# 32 because it gives more
```

```
features for curves and edges
    # layers.LeakyReLU(alpha=0.1),
    layers.Dropout(0.3),
    layers.MaxPooling2D((2,2)),# output size :(2,2,20)

    # layers.GaussianNoise(0.2),
    layers.Conv2D(num_classes, (2,2), activation='sigmoid'),# 32 because it
gives more features for curves and edges

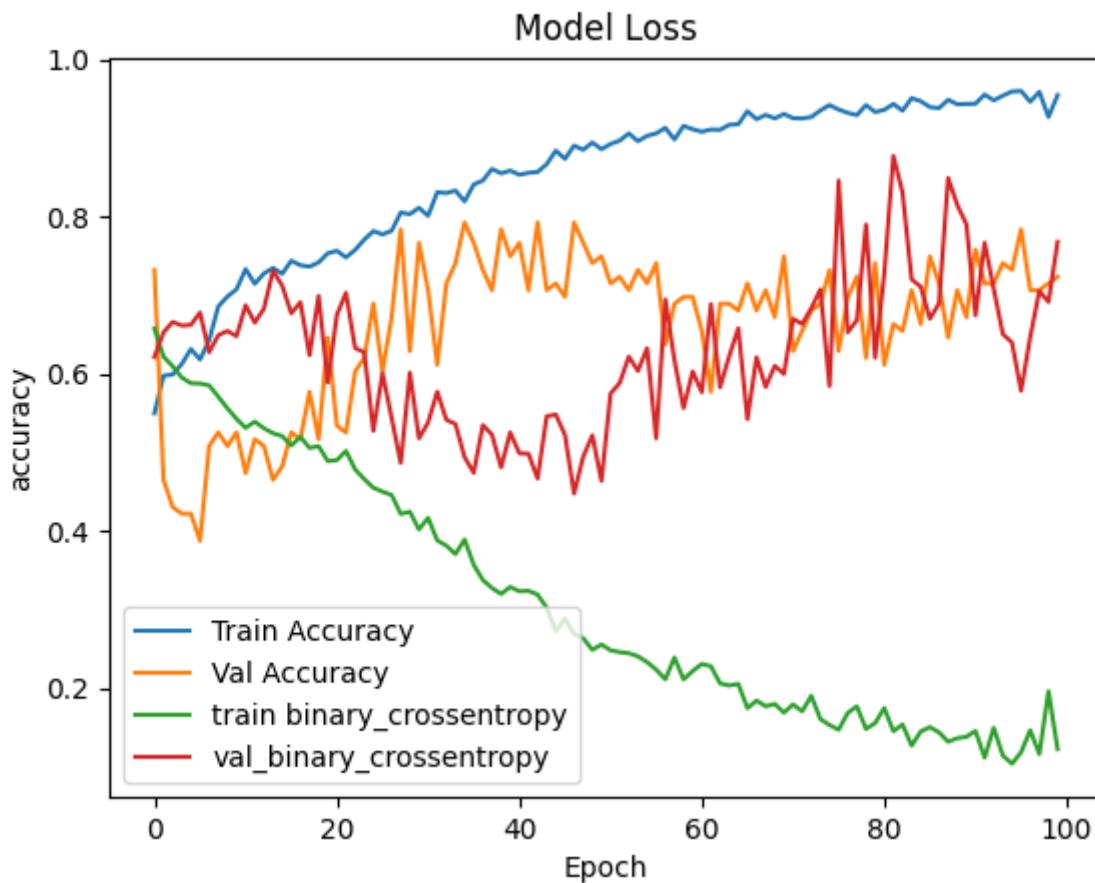
    layers.Flatten(),
```



zien dat de validation loss curved stijgt verder in vergelijking met een kleinere aantal kernels

Nu wil ik dense layers toevoegen om features te abstracten en generaliseerbaarheid te verbeteren.

we



ik

heb gemerkt dat bij images waar er veel ruis in zit het model gelijk aangeeft dat het een botbreuk is. Ik ga dus ruis toevoegen en ik start met een 0.05 gaussian noise en 0.3 dropout

```

layers.Conv2D(32 , (5, 5), input_shape=input_shape,activation = 'relu'),# 50
because it gives more features for curves and
    layers.Dropout(0.3),
    layers.MaxPooling2D((2, 2)),#output size :(30,30,20)

    layers.GaussianNoise(0.05),
    layers.Conv2D(32,(3,3), activation='relu'),
    layers.Dropout(0.3),
    layers.MaxPooling2D((2,2), ),#output: (14,14,30),

    layers.GaussianNoise(0.05),
    layers.Conv2D(50,(3,3), activation='relu'),
    layers.Dropout(0.3),
    layers.MaxPooling2D((2,2)),#output: (6,6,25)

    layers.GaussianNoise(0.05),
    layers.Conv2D(100, (3,3), activation='relu'),# 32 because it gives more
features for curves and edges
    # layers.LeakyReLU(alpha=0.1),
    layers.Dropout(0.3),
    layers.MaxPooling2D((2,2)),# output size :(2,2,20)

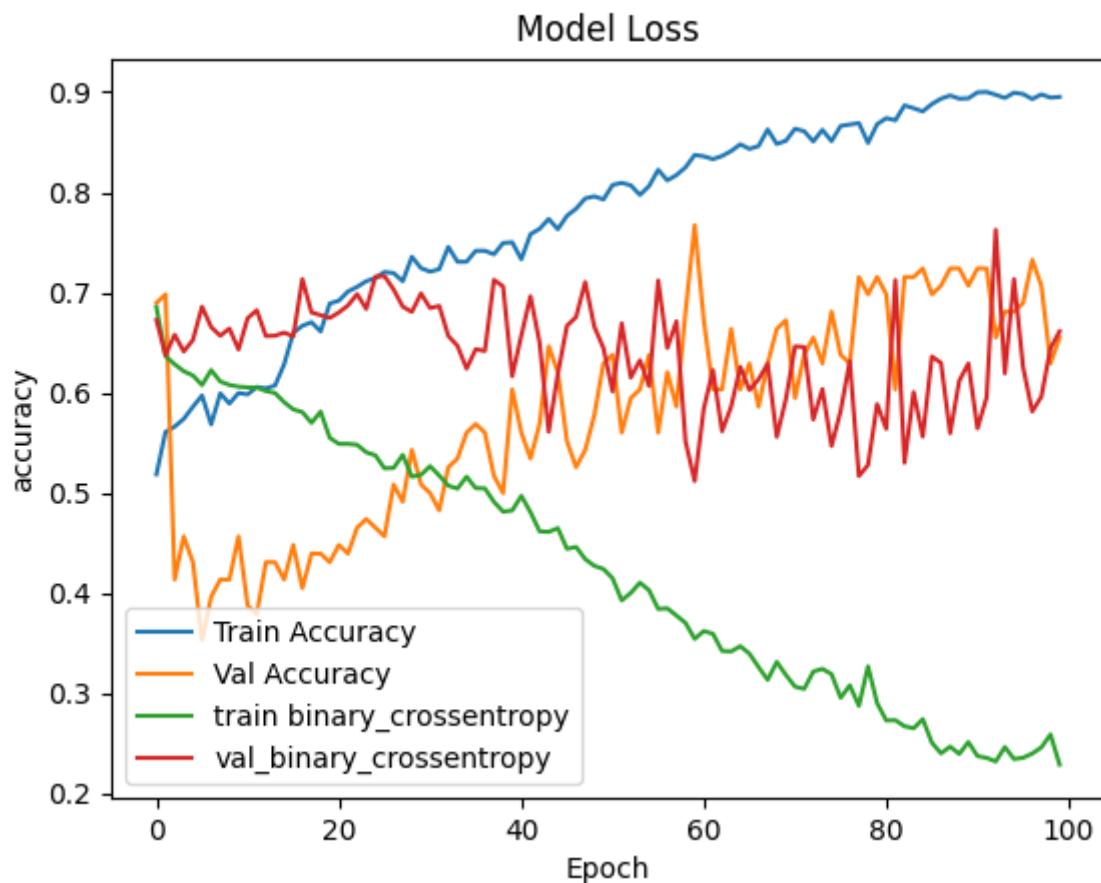
    # layers.GaussianNoise(0.2),
    # layers.Conv2D(num_classes, (2,2), activation='sigmoid'),# 32 because it

```

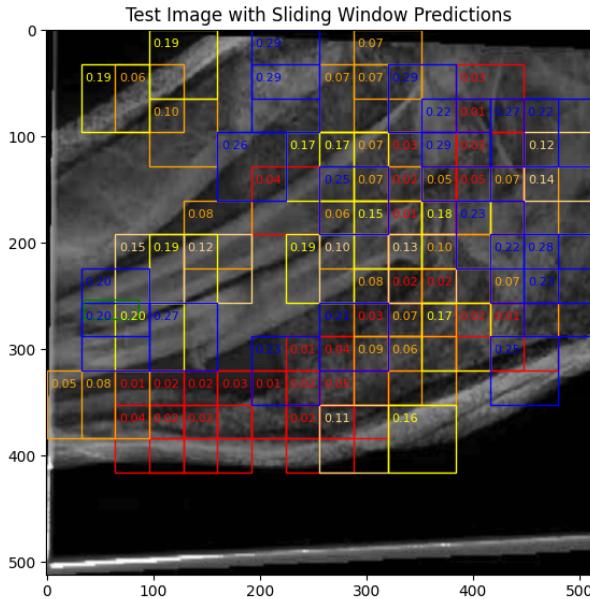
gives more features for curves and edges

```
layers.Flatten(),
# layers.Conv2D(15, (3,3), activation='relu', padding='same'),# 32 because
it gives more features for curves and edges
# layers.MaxPooling2D((2,2)),# output size :(4,4,10)
# layers.Flatten(),
layers.GaussianNoise(0.05),
layers.Dense(30, activation='relu'),
layers.LeakyReLU(alpha=0.1),
layers.Dropout(0.3),

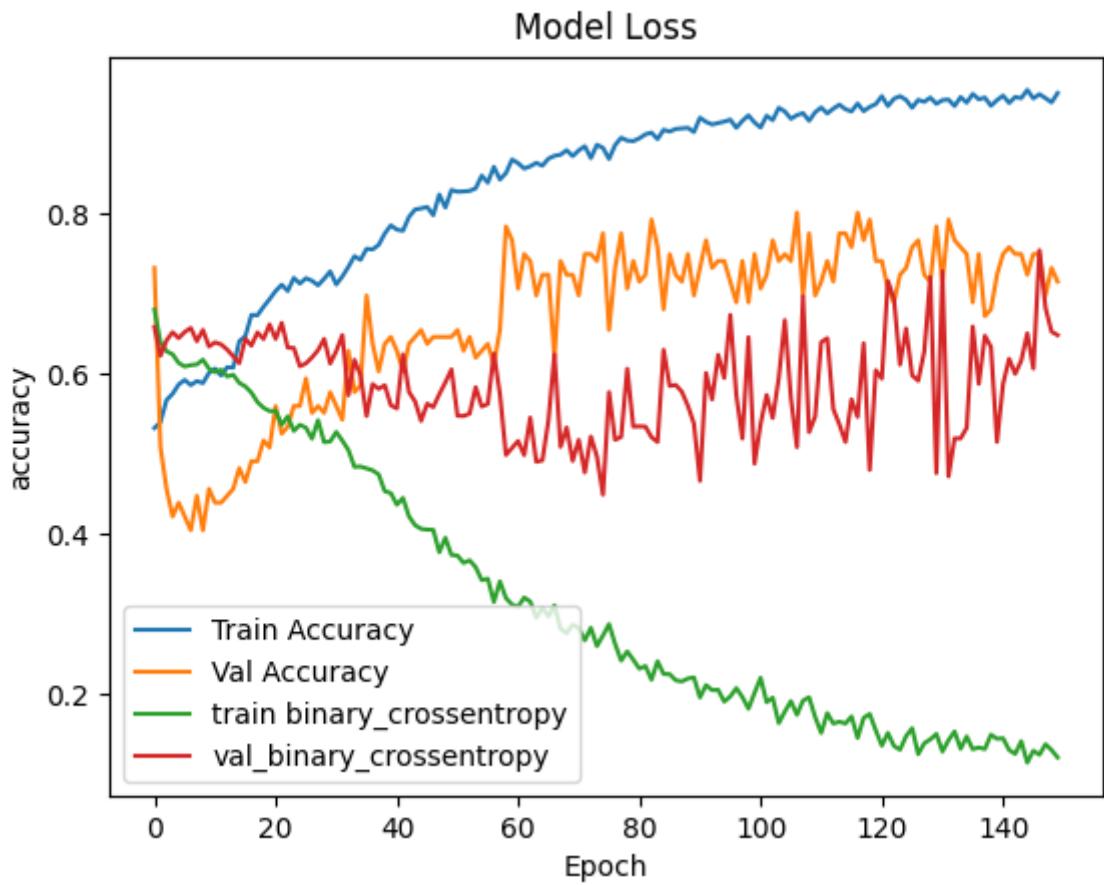
layers.GaussianNoise(0.01),
layers.Dense(15, activation='relu'),
layers.Dropout(0.3),
layers.Dense(num_classes,activation='sigmoid', name='output_layer'),
```



nu is hij iets minder gevoelig voor ruis



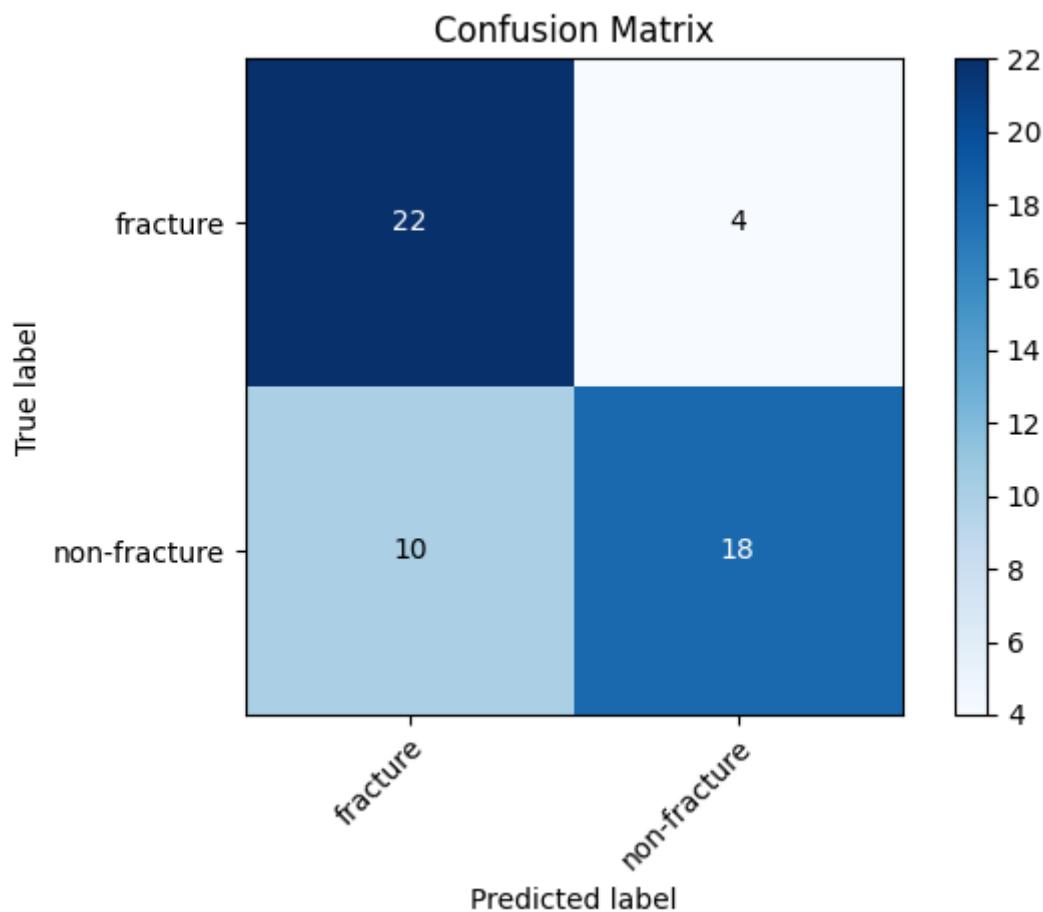
verder heb ik het aantal epochs gestegen naar 150 met de volgende resultaat. Ik dacht dat de tredn zou doorgaan stijgen, maar de val loss curve is gestabiliseerd smane met de training loss curve.



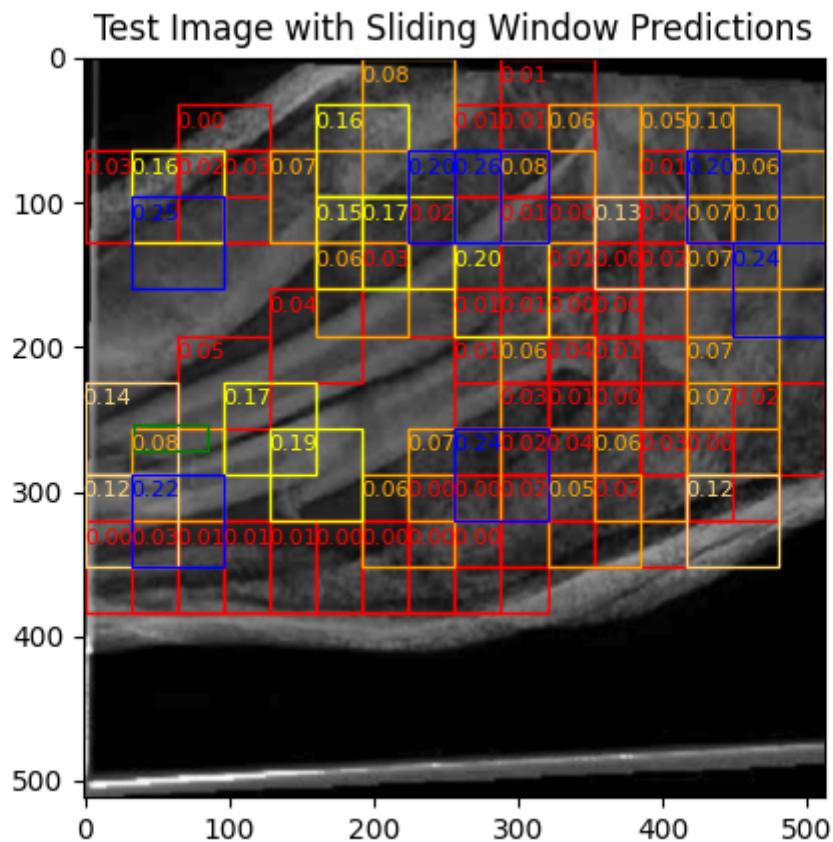
threshold vor het testen heb ik veranderd naar 0.3 omdat het model nu meer zeker is dat het en fractuur is,

de

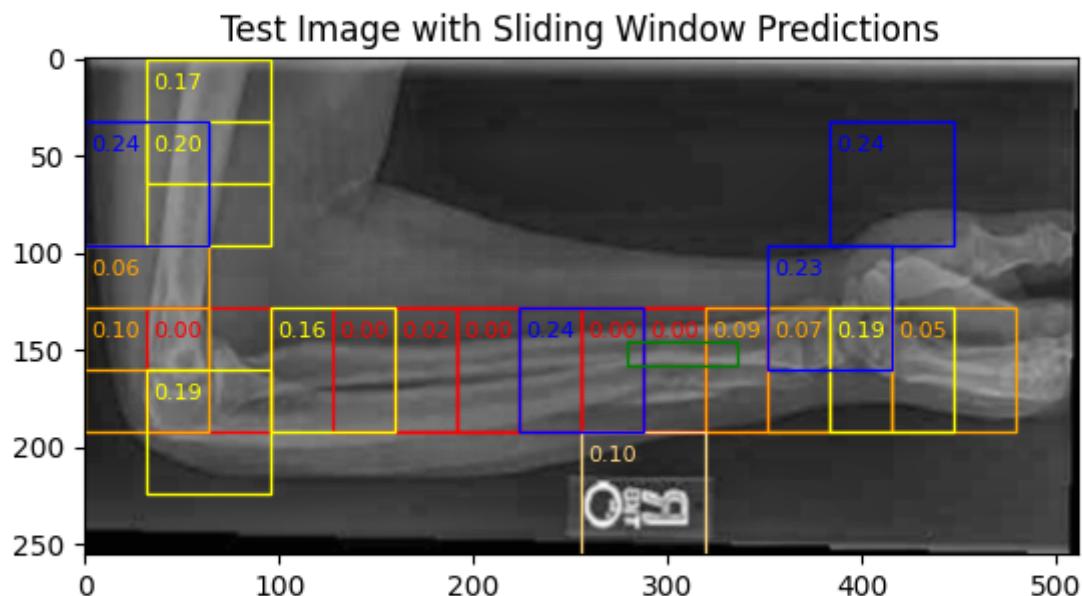
ook al is de fractuur niet aanwezig. Dit is te zien in de confusion matrix hieronder:



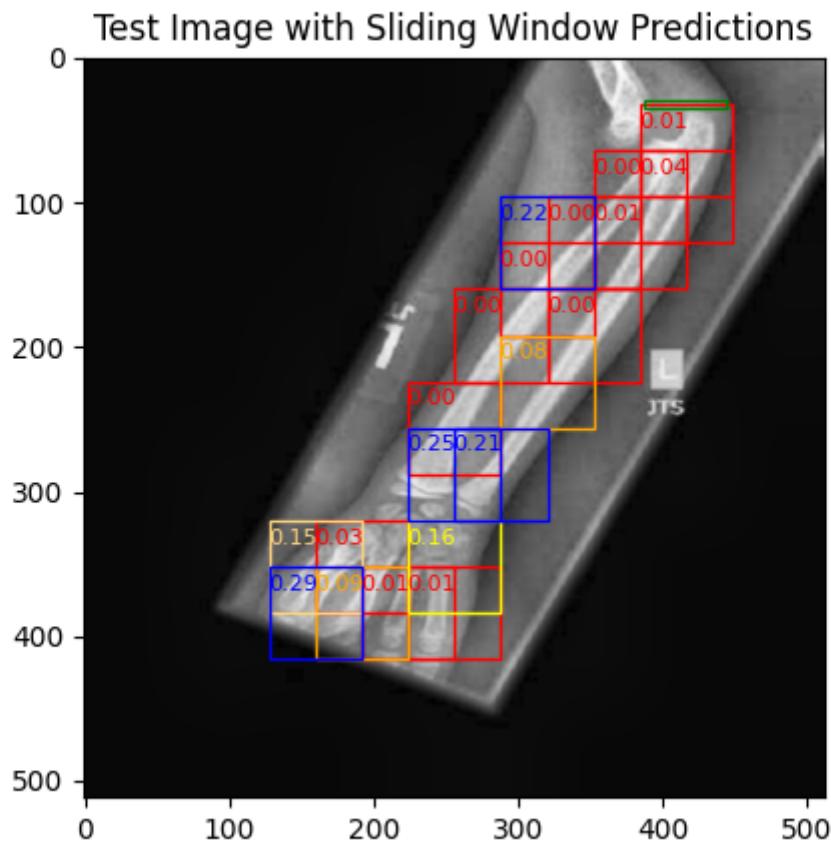
met een accuracy van 75% maar het wordt meer gevoelig voor



echter is de detectie van fractuur beter. Dit kunnen we merken aan de rode kleur waar er waarschijnlijk een fractuur zit.

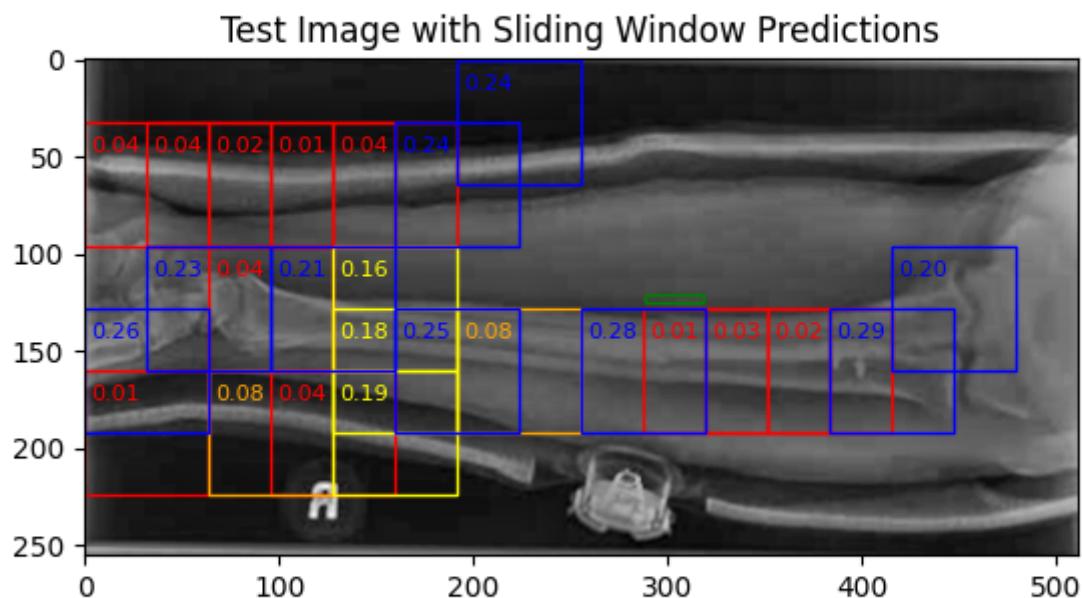


De afbeelding laat zien dat het model het verschil tussen de ruimte tussen de twee boten van de onderarm niet kan maken. Hiervoor zou ik een aparte klasse voor moeten maken die de onderarm botten als niet-fractuur labelt. Hieronder is er nog een voorbeeld van de detectie van onderarm. Maar we kunnen toch wel zien dat bij de rode pijl (waar de fractuur zit) het model wel aangeeft dat er een fractuur in zit. Hier is de structuur van beide boten van de onderarm niet zichtbaar



helaas blijft de gips een probleem voor het model. Maar toch kan het model wel de fractuur zien. Er zijn echter wel veel false positives, maar het is de eerste keer dat het model de fractuur detecteert op deze

afbeelding.



als
een laatste poging om een betere resultaat te krijgen heb ik geprobeerd met een verlaagde aantal kernels bij de laatste conv2D: layers.Conv2D(32 , (5, 5), input_shape=input_shape,activation = 'relu'),# 50 because it gives more features for curves and

```

layers.Dropout(0.3),
layers.MaxPooling2D((2, 2)),#output size :(30,30,20)

layers.GaussianNoise(0.05),
layers.Conv2D(32,(3,3), activation='relu'),
layers.Dropout(0.3),
layers.MaxPooling2D((2,2), ),#output: (14,14,30),

layers.GaussianNoise(0.1),
layers.Conv2D(50,(3,3), activation='relu'),
layers.Dropout(0.3),
layers.MaxPooling2D((2,2)),#output: (6,6,25)

layers.GaussianNoise(0.1),
layers.Conv2D(50, (3,3), activation='relu'),# 32 because it gives more
features for curves and edges
# layers.LeakyReLU(alpha=0.1),
layers.Dropout(0.3),
layers.MaxPooling2D((2,2)),# output size :(2,2,20)

# layers.GaussianNoise(0.2),

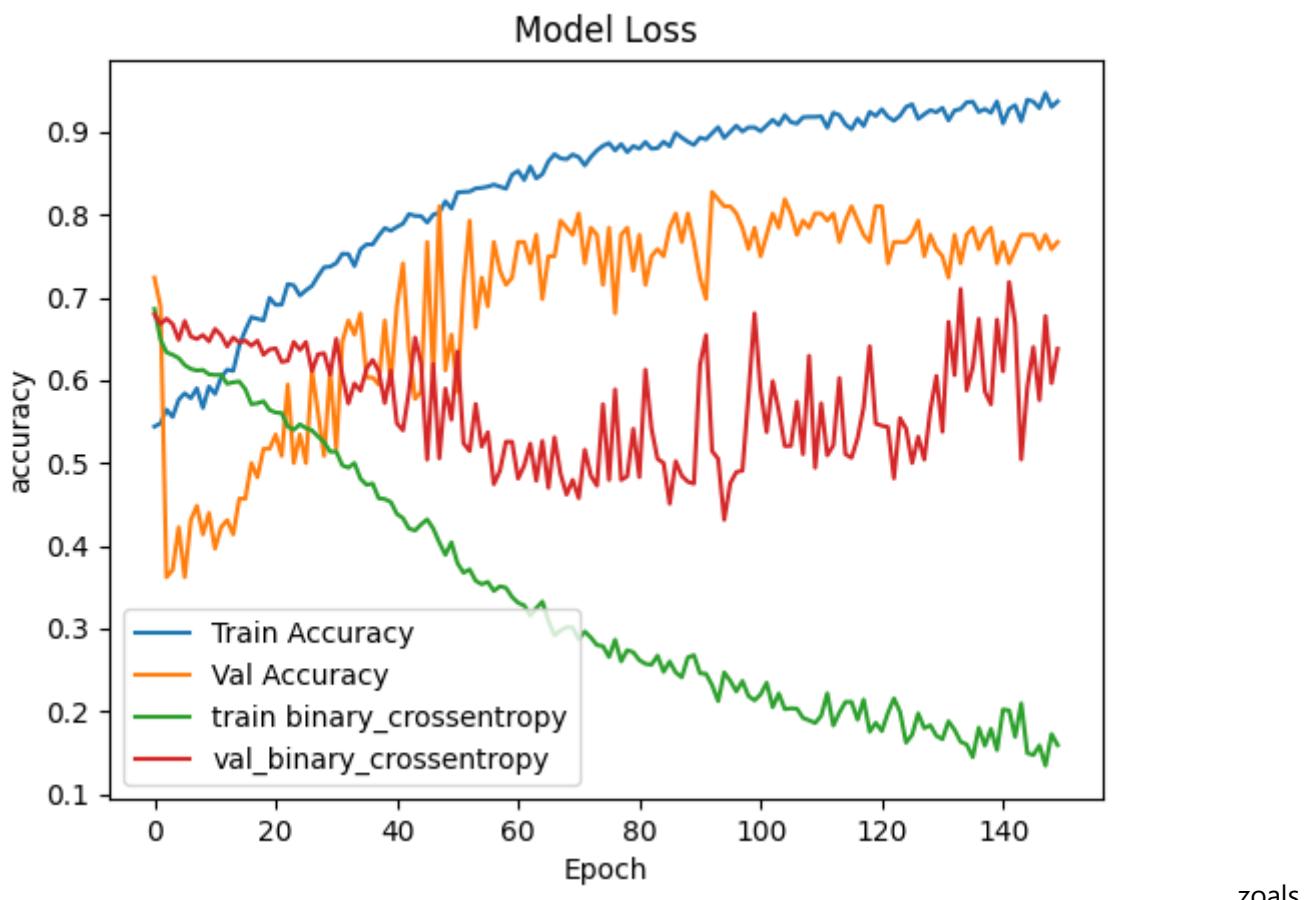
```

```
# layers.Conv2D(num_classes, (2,2), activation='sigmoid'),# 32 because it
gives more features for curves and edges

layers.Flatten(),
# layers.Conv2D(15, (3,3), activation='relu', padding='same'),# 32 because
it gives more features for curves and edges
# layers.MaxPooling2D((2,2)),# output size :(4,4,10)
# layers.Flatten(),
layers.GaussianNoise(0.1),
layers.Dense(35, activation='relu'),
layers.LeakyReLU(alpha=0.1),
layers.Dropout(0.3),

layers.GaussianNoise(0.1),
layers.Dense(15, activation='relu'),
layers.Dropout(0.3),
layers.Dense(num_classes,activation='sigmoid', name='output_layer'),
```

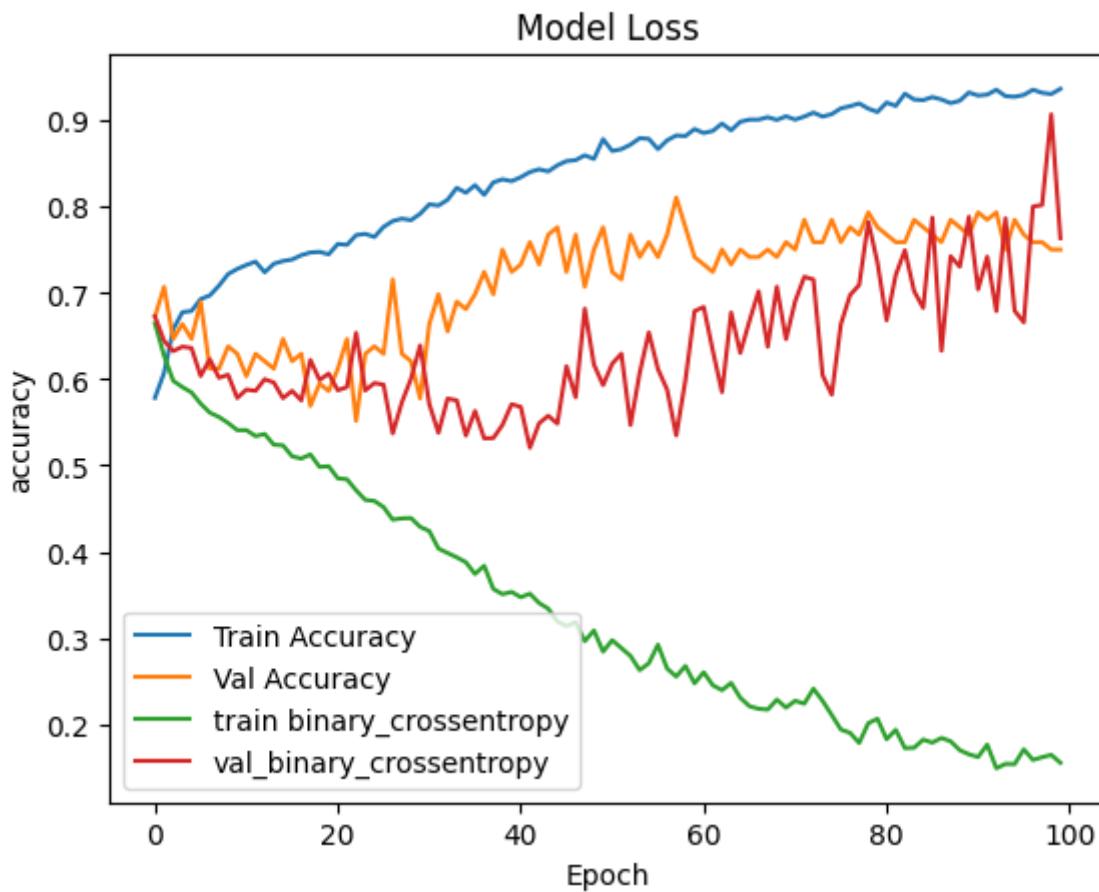
dit gaf de volgende loss curve:



zoals we kunnen zien is de val loss curve gestabiliseerd maar de validation accuracy is gestegen tot ongeveer 80%.

Dag 24:

gaussian noise op 0.1. We zien dat de val curve en de accuracy curve bij elkaar blijven. Dit betekent dat het model is op een limiet aangekomen waardoor een complexere model nodig is.



Ik heb met toevoegeing va dense layers geprobeerd :

```

layers.Conv2D(32 , (5, 5), input_shape=input_shape,activation = 'relu'),# 50
because it gives more features for curves and
    layers.Dropout(0.2),
    layers.MaxPooling2D((2, 2)),#output size :(30,30,20)
    layers.GaussianNoise(0.1),
    layers.Conv2D(32,(3,3), activation='relu'),
    layers.Dropout(0.2),
    layers.MaxPooling2D((2,2), ),#output: (14,14,30),
    layers.GaussianNoise(0.1),
    layers.Conv2D(64,(3,3), activation='relu'),
    layers.Dropout(0.2),
    layers.MaxPooling2D((2,2)),#output: (6,6,25)

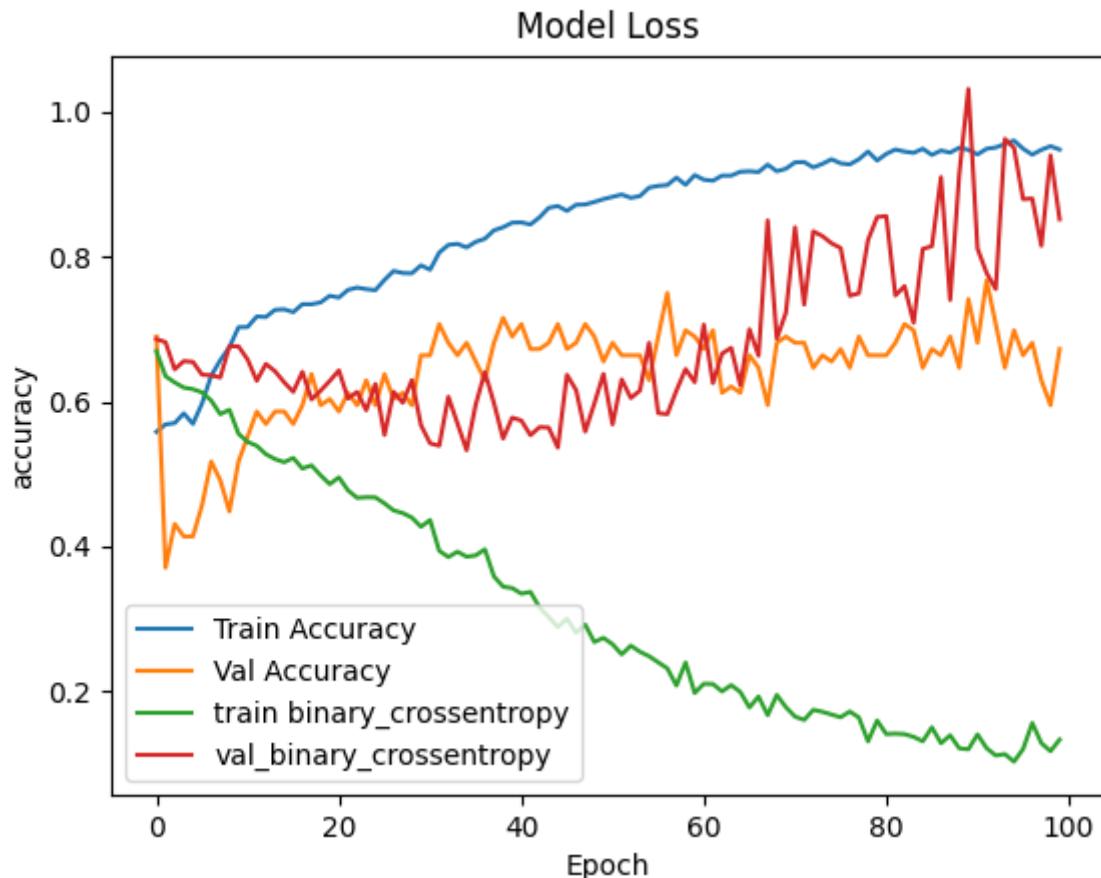
    layers.GaussianNoise(0.1),
    layers.Conv2D(128, (3,3), activation='relu'),#
    layers.Dropout(0.2),
    layers.MaxPooling2D((2,2)),# output size :(2,2,128)

    layers.Flatten(),# output size 512
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),

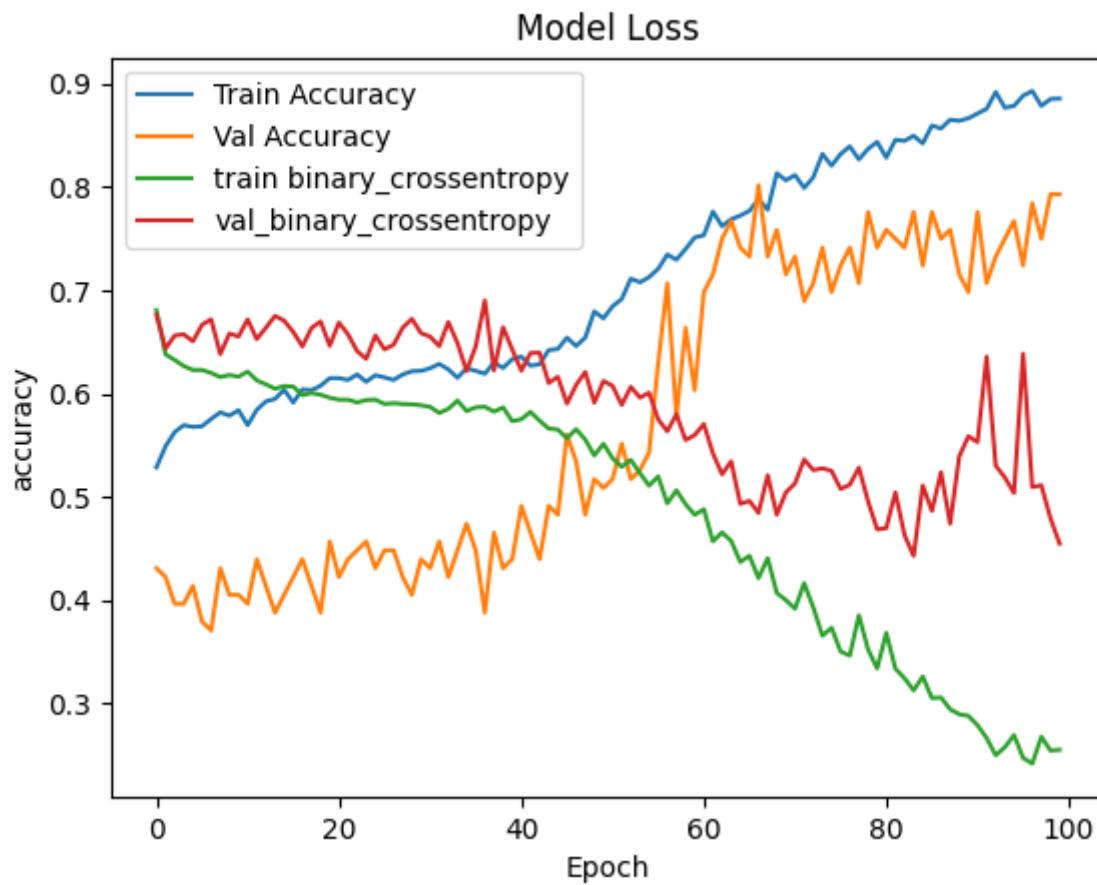
```

```
layers.Dense(64, activation='relu'),  
layers.Dropout(0.2),  
layers.Dense(num_classes, activation='sigmoid')
```

Dit gaf de volgende loss curve:

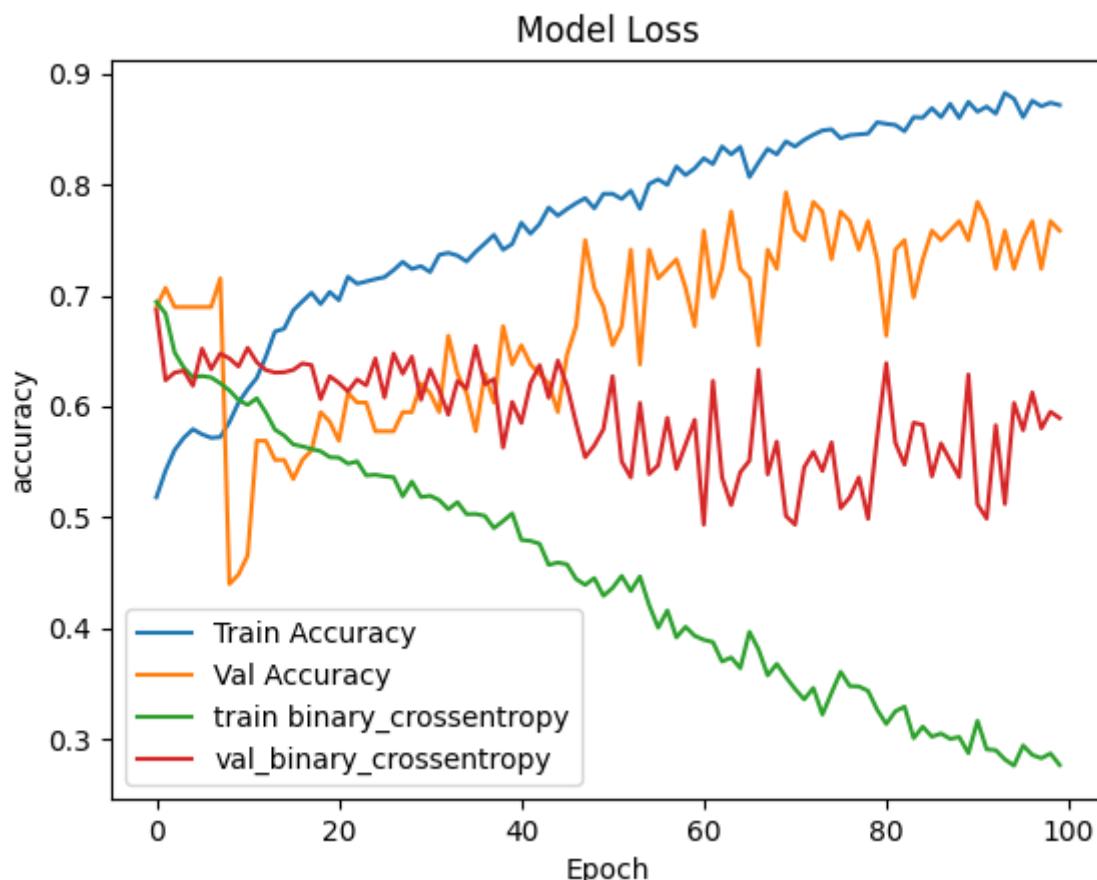


ik heb nu het model getraind met minder neuronen per dense layer, namelijk 64, 32 en 16



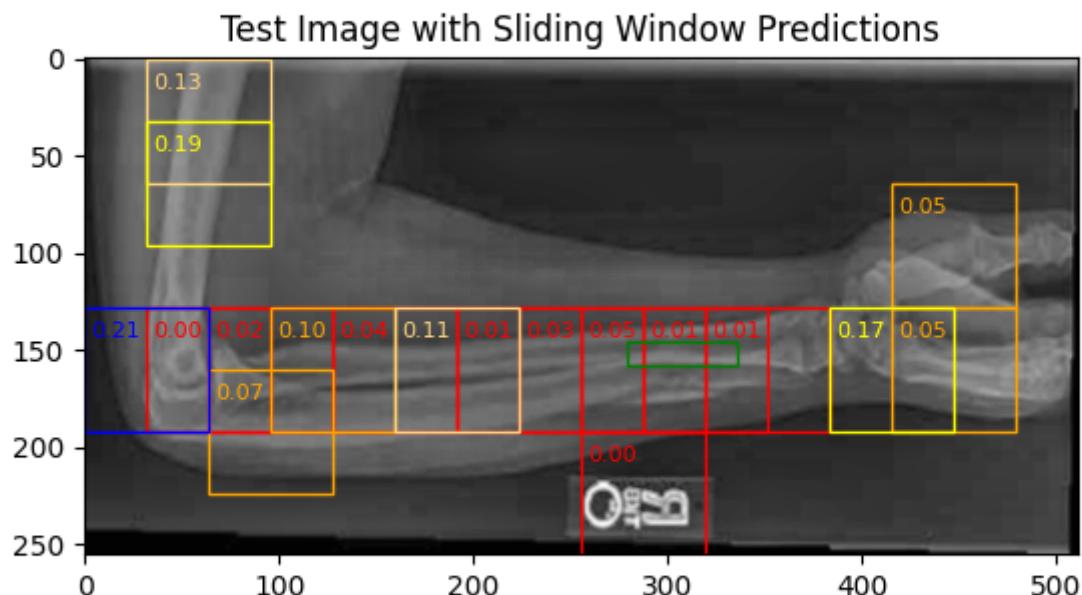
het

model heeft een betere accuraatheid in voor het testen 85%. ik heb de dense layers verlaagd naar 32, 16, en 8 en krijg deze loss curve:



We zien dat de divergence van de val loss en training loss curve begint bij epoch 65. Ik had verwacht dat de vermindering van het aantal neuronen een betere uitkomst zou brengen omdat er waarschijnlijk minder overfitting zou opleveren. Het is alsof het model probeert vanaf de 65 epoch andere paden te proberen om algemenere patronen te vinden, maar het is niet in staat vanwege een beperking in de dataset voor variatie. We zien dat de train accuracy blijft toenemen. Hierdoor heb ik op het aantal neuronen terug gebracht naar 64 -> 32 -> 16 en de dropout naar 0.2.

Het idee is dat het model leert dat botbreuken geen gladde randen zijn maar juist onregelmatige en scherpe randen zijn. Deze kenmerken worden al in de vroege convolutielagen gedetecteerd, omdat deze filters vooral gericht zijn op het oppikken van randen en textuurvariaties. Naarmate de data verder door het netwerk gaat, worden deze eenvoudige kenmerken gecombineerd tot complexere patronen, zoals de specifieke vorm en locatie van de breuk in de afbeelding. Uiteindelijk wordt in de dense layer op basis van deze abstracte en samengestelde kenmerken de beslissing genomen of het daadwerkelijk om een botbreuk gaat. Vervolgens heb ik sliding window methode weer toegepast en gelijkweg krijg ik een betere resultaat:



Hij lijkt nu beter te generaliseren, maar toch blijft de variatie in de voorspelligen nog steeds aanwezig. dat kunnen we zien in het plaatje bij de letters die het model als fractuur labelt.