جامعة دمشق

كلية الهندسة المعلوماتية

اختصاص الذكاء الصنعي

تقرير مشروع الشبكات العصبونية

أعداد الطلاب :

عبد الرؤوف حسحس

محمد عثمان دياربكرلي

الفهرس

Contents

[توصيف الفكرة : 3](#_Toc155146153)

[توصيف الداتا سيت المستخدمة : 3](#_Toc155146154)

[طريقة التدريب الأولى : 4](#_Toc155146155)

[ بنية الشبكة 4](#_Toc155146156)

[ التدريب: 4](#_Toc155146157)

[o دوال التفعيل 4](#_Toc155146158)

[o فئة الشبكة العصبية (NeuralNetwork) 4](#_Toc155146159)

[o آلية التدريب 5](#_Toc155146160)

[ شرح خوارزميات تعديل الأوزان المستخدمة 6](#_Toc155146161)

[o الانتشار العكسي التقليدي (Backpropagation) 6](#_Toc155146162)

[o الانتشار العكسي المرن (Resilient Backpropagation، اختصارًا Rprop) 7](#_Toc155146163)

[o الشبكات العصبية البيزية (Bayesian Neural Networks) 9](#_Toc155146164)

[طريقة التدريب الثانية : 10](#_Toc155146165)

[النتائج والتقييم لكال الشبكتين: 10](#_Toc155146166)

[الشبكة الأولى 10](#_Toc155146167)

[الشبكة الثانية 11](#_Toc155146168)

[التقييم 11](#_Toc155146169)

[مقارنة نتائج التجارب 12](#_Toc155146170)

[12](#_Toc155146171)

[Underfitting 12](#_Toc155146172)

[Overfitting 12](#_Toc155146173)

[االمصادر 13](#_Toc155146174)

Intel image classification

# توصيف الفكرة :

* اسم الفكرة المختارة: Intel image classification .
* رقم الفكرة داخل جدول الأفكار: الفكرة رقم تسعة (9) .
* الهدف من الفكرة :عمل تصنيف لمجموعة من الصورة مكونة من عدة أصناف (muticlassification).

# توصيف الداتا سيت المستخدمة :

* المصدر: لقد تم تحميل الداتا من موقع . kaggle
* نوع البيانات: البيانات مكونة من مجموعة من الصورة مقسمة إلى ستة أصناف (buildings -forest -glacier -mountain - sea-street ) .
* حجم البيانات: لدينا الداتا مكونة من 17034 صورة مقسمة إلى 14034 صورة من أجل التدريب و 3000 من أجل الاختبار .
* العمليات المطبقة على الداتا قبل بدء التدريب:أولاً قمنا بقرائة الصور وتحويلها إلى صور بتدرج رمادي وعمل تغير لحجم الصورة من أجل تقليل المزايا للصورة و توحيدة و من ثما حددنا نوع المصفوفة المكونة للصور ب float32 ومن ثما قسمنا قيم المصفوفة على 255 من أجل جعل القيم بين الواحد والصفر و قمنا بتحويل جدول ال labels إلى One Hot Encoder .

# طريقة التدريب الأولى :

* بنية الشبكة: لقد قمنا ببناء شبكة مكونة من طبقة دخل و طبقتين مخفيتين و طبقة خرج، حيث تم تهيئة أوزان الطبقة الأولى بعدد الميزات في البيانات الإدخال و 128 خلية عصبية للطبقة الأولى أما بالنسبة للطبقة الثانية تم تهيئة الأوزان ب 128 عدد خلايا الطبقة السابقة و 64 عدد خلايا الطبقة الحالية أما بالنسبة للطبقة الثالثة تم تهيئة الأوزان ب 64 عدد خلايا الطبقة السابقة و 32 عدد خلايا الطبقة الحالية أما بالنسبة لطبقة الخرج تم تهيئة الأوزان ب 32 عدد خلايا الطبقة السابقة و 6 عدد خلايا الطبقة الحالية وهي أيضاً عدد الأصناف لدينا وقد قمنا بضرب جميع مصفوفات الأوزان ب 0.01 من أجل تكبير قيم الأوزان .

## التدريب:

* + دوال التفعيل :
    - Sigmoid (السيجمويد): دالة تحول أي قيمة إلى قيمة بين 0 و 1. مفيدة للتصنيف الثنائي.
    - Softmax (السوفتماكس): تُستخدم للتصنيف متعدد الفئات، حيث تحول النتائج الخطية إلى احتمالات تجمع إلى واحد.
    - ReLU (وحدة الخطية المعدلة): تُستخدم عادة في الطبقات المخفية وتسمح للنماذج بأخذ اللاخطية في الحسبان.
    - المشتقات:
      * مشتق ال Sigmoidومشتق ال ReLUضرورية لحساب التدرجات أثناء الأنتشار العكسي.
  + فئة الشبكة العصبية (NeuralNetwork):
    - التهيئة (init): تبدأ الأوزان والتحيزات والمتغيرات الضرورية الأخرى.
    - التنبؤ (predict): بالنظر إلى البيانات المدخلة، تتوقع الفئة عن طريق التمرير عبر الشبكة واختيار الاحتمال الأعلى.
    - الدقة (accuracy): يقارن بين القيم المتوقعة والفعلية لحساب دقة النموذج.
    - الانتشار الأمامي (feedforward): يمرر البيانات خطيًا عبر طبقات الشبكة باستخدام الأوزان والتحيزات ويطبق دوال التفعيل.
    - الانتروبيا التبادلية (cross\_entropy): دالة الخسارة المستخدمة لقياس عدم التشابه بين الاحتمالات المتوقعة والتوزيع الفعلي.
    - الانتشار العكسي (backprop): يضبط الأوزان والتحيزات في الاتجاه المعاكس للتدرج لتقليل الخسارة. يستخدم مشتقات دوال التفعيل.
    - الانتشار المرن (backprop\_rprop): بديل للانحدار التدرجي التقليدي، يركز على اتجاه التدرج وليس حجمه، ويضبط تحديث كل وزن بشكل مستقل.
    - تحديث الوزن البيزي (bayesian\_weight\_update): يدمج مصطلح الضوضاء في تحديثات الوزن، مقلدًا أخذ العينات اللاحقة البيزية.
    - التدريب البيزي (train\_bayesian): دالة تدريب تحدث الأوزان باستخدام النهج البيزي لكل دفعة من البيانات.
  + آلية التدريب:
    - التهيئة: قبل التدريب، يتم تهيئة الأوزان عشوائيًا قريبة من الصفر ولكن ليس بالضبط الصفر.
    - الانتشار الأمامي: لكل مثال تدريب، يتم تمرير المدخلات عبر طبقات الشبكة ، تطبيق دوال التفعيل على المخرجات الخطية. ثم، تُحول القيم بواسطة الدوال المختارة (سيجمويد أو ريلو) إلى قيم تكون إما احتمالات في حالة السوفتماكس أو قيم مفعلة في الطبقات المخفية.
    - الانتشار العكسي: بعد حساب الخطأ من الناتج النهائي، يتم انتشار هذا الخطأ إلى الوراء في الشبكة، مما يؤدي إلى تحديث الأوزان والتحيزات. يستخدم هذا الجزء مشتقات دوال التفعيل لحساب التغييرات اللازمة في كل طبقة.
    - التحديث الوزني: يمكن أن يتم عبر عدة طرق مثل الانتشار العكسي التقليدي، الانتشار المرن (RProp)، أو التحديثات البيزية، حيث يختار المطور آلية التحديث بناءً على متطلبات النموذج وخصائص البيانات.
* شرح خوارزميات تعديل الأوزان المستخدمة:
  + الانتشار العكسي التقليدي (Backpropagation): خوارزمية الانتشار العكسي تعمل بنشر الخطأ من الطبقة الخارجية إلى الطبقات الداخلية في الشبكة العصبية بشكل عكسي، حيث يتم استخدام مشتقات دالة الخسارة لتحديث الأوزان والتحيزات بطريقة تقلل من الخطأ الكلي. تُعد هذه العملية حيوية لتدريب الشبكات العصبية، حيث تتكرر لعدة مرات حتى يتم تحقيق الدقة المطلوبة أو حتى يصبح التحسين ضئيلاً.

def backprop\_rprop(self):

    # حساب التدرجات

    error\_output = (self.y - self.output)

    d\_weights4 = np.dot(self.layer3.T, error\_output)

    error\_layer3 = np.dot(error\_output, self.weights4.T) \* relu\_derivative(self.layer3)

    d\_weights3 = np.dot(self.layer2.T, error\_layer3)

    error\_layer2 = np.dot(error\_layer3, self.weights3.T) \* relu\_derivative(self.layer2)

    d\_weights2 = np.dot(self.layer1.T, error\_layer2)

    error\_layer1 = np.dot(error\_layer2, self.weights2.T) \* relu\_derivative(self.layer1)

    d\_weights1 = np.dot(self.input.T, error\_layer1)

error\_output = (self.y - self.output): هنا يُحسب الخطأ في الطبقة النهائية من الشبكة. self.y هو المخرج المستهدف (القيم الصحيحة) و self.output هو الإخراج الفعلي الذي تم توليده من الشبكة. هذا الخطأ يُستخدم كأساس لتحديث الأوزان في الطبقة الأخيرة.

d\_weights4 = np.dot(self.layer3.T, error\_output): يُحسب هذا الجزء التغيير المطلوب للأوزان الربط بين الطبقة الثالثة والطبقة النهائية. يتم ذلك بضرب ترانسبوز الطبقة الثالثة في خطأ الإخراج.

يُحسب الأخطاء والتغييرات المتعلقة بالأوزان للطبقات الأخرى بطريقة مشابهة:

error\_layer3 = np.dot(error\_output, self.weights4.T) \* relu\_derivative(self.layer3): يحسب الخطأ في الطبقة الثالثة باستخدام الخطأ في الإخراج وأوزان الطبقة النهائية. يضرب في مشتق ReLU للحسابات المتعلقة باللاخطية.

d\_weights3 = np.dot(self.layer2.T, error\_layer3): يحسب التغيير المطلوب للأوزان بين الطبقة الثانية والطبقة الثالثة.

error\_layer2 = np.dot(error\_layer3, self.weights3.T) \* relu\_derivative(self.layer2): يحسب الخطأ في الطبقة الثانية باستخدام الخطأ في الطبقة الثالثة وأوزان الطبقة الثانية. يضرب في مشتق ReLU.

d\_weights2 = np.dot(self.layer1.T, error\_layer2): يحسب التغيير المطلوب للأوزان بين الطبقة الأولى والطبقة الثانية.

error\_layer1 = np.dot(error\_layer2, self.weights2.T) \* relu\_derivative(self.layer1): يحسب الخطأ في الطبقة الأولى باستخدام الخطأ في الطبقة الثانية وأوزان الطبقة الأولى. يضرب في مشتق ReLU.

d\_weights1 = np.dot(self.input.T, error\_layer1): يحسب التغيير المطلوب للأوزان بين الطبقة الداخلية والطبقة الأولى.

يتم تحديث الأوزان لكل طبقة باستخدام معدل التعلم (l\_rate):

self.weights1 += l\_rate \* d\_weights1

self.weights2 += l\_rate \* d\_weights2

self.weights3 += l\_rate \* d\_weights3

self.weights4 += l\_rate \* d\_weights4

* + الانتشار العكسي المرن (Resilient Backpropagation، اختصارًا Rprop): يعد تحسينًا لخوارزمية الانتشار العكسي التقليدية وهو مصمم للتغلب على مشاكل الحساسية لمعدل التعلم ولتسريع عملية التقارب. بدلاً من تحديث الأوزان بناءً على مقدار التدرج نفسه، تعتمد Rprop فقط على الإشارة (الاتجاه) للتدرج. في كل تكرار، يتم تحديث كل وزن بخطوة ثابتة نحو التقليل من الخطأ، حيث يمكن زيادة أو تقليل هذه الخطوة بناءً على التغيرات المتسقة في اتجاه التدرج لكل وزن عبر التكرارات.إذا كان التغيير في الاتجاه لتدرج وزن ما متسقًا عبر التكرارات، فإن خطوة التحديث لذلك الوزن تزداد لتسريع التعلم. وعلى العكس، إذا كان التغيير في الاتجاه غير متسق (أي يتغير الاتجاه من تكرار لآخر)، تقل الخطوة لمنع التقلبات الكبيرة ولتحسين الاستقرار. هذا يجعل Rprop فعالاً بشكل خاص لشبكات الحجم الكبير والمشكلات التي تحتوي على سطوح خطأ معقدة.
* def backprop\_rprop(self):
* # حساب التدرجات
* error\_output = (self.y - self.output)
* d\_weights4 = np.dot(self.layer3.T, error\_output)
* error\_layer3 = np.dot(error\_output, self.weights4.T) \* relu\_derivative(self.layer3)
* d\_weights3 = np.dot(self.layer2.T, error\_layer3)
* error\_layer2 = np.dot(error\_layer3, self.weights3.T) \* relu\_derivative(self.layer2)
* d\_weights2 = np.dot(self.layer1.T, error\_layer2)
* error\_layer1 = np.dot(error\_layer2, self.weights2.T) \* relu\_derivative(self.layer1)
* d\_weights1 = np.dot(self.input.T, error\_layer1)
* # تطبيق RPROP
* for i, (prev\_grad, step\_size, d\_weight, weight) in enumerate(zip(self.prev\_gradients, self.step\_sizes, [d\_weights1, d\_weights2, d\_weights3, d\_weights4], [self.weights1, self.weights2, self.weights3, self.weights4])):
* sign = np.sign(d\_weight \* prev\_grad)
* step\_size[sign > 0] \*= self.eta\_plus
* step\_size[sign < 0] \*= self.eta\_minus
* step\_size = np.clip(step\_size, self.min\_step, self.max\_step)
* weight\_update = np.sign(d\_weight) \* step\_size
* weight += weight\_update
* prev\_grad[sign >= 0] = d\_weight[sign >= 0]
* self.prev\_gradients[i] = prev\_grad
* self.step\_sizes[i] = step\_size

قمنا بحساب التدرجات مثل الخوارزمية السابقة

يتم تكرار عملية التحديث لكل وزن في الشبكة (self.weights1, self.weights2, self.weights3, self.weights4) باستخدام معلومات مُخزَّنة من التدريبات السابقة.

zip(self.prev\_gradients, self.step\_sizes, [d\_weights1, d\_weights2, d\_weights3, d\_weights4], [self.weights1, self.weights2, self.weights3, self.weights4]): يقوم بدمج معلومات مُخزَّنة حول السابق لكل طبقة مع الوزن الحالي وتغيير الوزن لتكوين أربعة أزواج (previous gradient, step size, d\_weight, weight).

يتم استخدام المتغير sign لتحديد اتجاه تغيير الوزن. إذا كانت الاشتقاقية الحالية d\_weight والاشتقاقية السابقة prev\_grad للوزن نفس الاتجاه، يتم تحديث حجم الخطوة (step\_size). إذا كانت في اتجاه معاكس، يتم تقليل حجم الخطوة.

step\_size[sign > 0] \*= self.eta\_plus: يعدل حجم الخطوة بناءً على اتجاه الاشتقاقية. إذا كانت في الاتجاه الإيجابي، يتم ضرب حجم الخطوة في self.eta\_plus لزيادته.

step\_size[sign < 0] \*= self.eta\_minus: يعدل حجم الخطوة بناءً على اتجاه الاشتقاقية. إذا كانت في الاتجاه السالب، يتم ضرب حجم الخطوة في self.eta\_minus لتقليله.

step\_size = np.clip(step\_size, self.min\_step, self.max\_step): يقوم بتقليص حجم الخطوة إذا تجاوزت الحد الأدنى أو الحد الأقصى المحدد.

weight\_update = np.sign(d\_weight) \* step\_size: يحسب تغيير الوزن الذي سيتم إضافته إلى الوزن الحالي. يُقسم حجم الخطوة إلى اتجاه التغيير.

weight += weight\_update: يُطبق التحديث النهائي للوزن باستخدام تغيير الوزن المحسوب.

prev\_grad[sign >= 0] = d\_weight[sign >= 0]: يُحدث الاشتقاق السابق إلى قيمة الاشتقاق الحالي إذا كانت في نفس الاتجاه.

self.prev\_gradients[i] = prev\_grad: يُحدث المتغير الذي يُخزِّن الاشتقاقات السابقة للاستخدام في التدريبات اللاحقة.

self.step\_sizes[i] = step\_size: يُحدث المتغير الذي يُخزِّن أحجام الخطوات للاستخدام في التدريبات اللاحقة.

* + الشبكات العصبية البيزية (Bayesian Neural Networks) : هي نوع من الشبكات العصبية التي تدمج مبادئ الاحتمالات البيزية في تعلم وتحسين الشبكات العصبية التقليدية. تختلف عن الشبكات العصبية التقليدية بأنها لا تقدم ناتجًا نهائيًا ثابتًا فقط، بل تقدم توزيع احتمالي لكل وزن داخل الشبكة، مما يعني أنها توفر قياسًا للتأكد والشك في التنبؤات التي تقوم بها.الفائدة الرئيسية للشبكات العصبية البيزية هي قدرتها على قياس وتحسين عدم اليقين في التنبؤات، مما يجعلها أداة قوية في الحالات التي يكون فيها البيانات نادرة أو متقلبة. كما أنها تمكن من إجراء تحديثات مستمرة للنموذج بطريقة مبنية على الاستدلال البيزي، مما يجعلها مرنة للتعلم من بيانات جديدة بشكل مستمر.
* def bayesian\_weight\_update(self, weight, gradient, learning\_rate):
* noise\_variance = 0.01
* bayesian\_noise = np.random.normal(0, noise\_variance, weight.shape)
* updated\_weight = weight + learning\_rate \* gradient + bayesian\_noise
* return updated\_weight

noise\_variance = 0.01: يُعين انحراف معياري لتحديد قوة ضوضاء بايزية التي ستتم إضافتها إلى الوزن.

bayesian\_noise = np.random.normal(0, noise\_variance, weight.shape): يُنشئ ضوضاء بايزية باستخدام توزيع طبيعي. يتم توليد أرقام عشوائية مع متوسط صفر وانحراف معياري مُعطى (noise\_variance). الحجم (shape) يكون مماثلًا للوزن الذي يتم تحديثه.

updated\_weight = weight + learning\_rate \* gradient + bayesian\_noise: يُحسب التحديث البايزي للوزن. يُضاف للوزن الحالي المضروب في معدل التعلم (learning\_rate) في اتجاه الجرادينت، ويُضاف له الضوضاء بايزية.

return updated\_weight: تُرجع الوزن المحدث بعد تنفيذ عملية التحديث البايزي.

# طريقة التدريب الثانية :

* Sequential(): هذا يُنشئ نموذجًا عصبيًا تسلسليًا، وهو نموذج يتم بناءه طبقة بطبقة.
* model.add(Dense(1024, input\_dim=images.shape[1], activation='relu')): يتم إضافة طبقة كاملة الاتصال (Dense) بعدد وحدات 1024، وتُستخدم وظيفة التنشيط ReLU. الطبقة الأولى تأخذ البيانات الواردة (images) كإدخال، ويتم تحديد البُعد الإدخالي بواسطة input\_dim.
* model.add(Dense(512, activation='relu')): تمثل الطبقة الثانية طبقة كاملة الاتصال أيضًا، لكنها تحتوي على 512 وحدة مع تنشيط ReLU.
* model.add(Dense(512, activation='relu')): تمثل الطبقة الثالثة طبقة كاملة الاتصال أيضًا بعدد 512 وحدة مع تنشيط ReLU.
* model.add(Dropout(0.2)): هذه الطبقة هي طبقة إسقاط (Dropout)، والتي تُستخدم لتجنب الفرط (Overfitting). تقوم بإسقاط 20٪ من الوحدات بشكل عشوائي خلال التدريب.
* يتبع النموذج بعد ذلك مجموعة من الطبقات المماثلة، مثل الطبقات Dense و Dropout، مما يساعد في استخراج وتعلم المزيد من المميزات من البيانات.
* model.add(Dense(len(label\_to\_number), activation='softmax')): آخر طبقة تمثل طبقة كاملة الاتصال تحتوي على عدد من الوحدات يتناسب مع عدد الفئات المختلفة في المشكلة (يُفضل استخدام softmax للمشكلات الفئوية). هذه الطبقة تُعطي إخراجًا يُفسر كاحتمالات لكل فئة.
* model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']): يُعد هذا السطر من التعليمات لتكوين النموذج. يُستخدم مُحدد الخسارة 'categorical\_crossentropy' لمشكلات التصنيف الفئوي، ويتم تحديد 'adam' كمحدد لعملية تحديث الأوزان. يتم استخدام المقياس 'accuracy' لتقدير أداء النموذج.
* model.summary(): يُعرض ملخص للنموذج، بما في ذلك عدد الوحدات في كل طبقة وعدد الباراميترات الإجمالي للنموذج. يُفيد هذا في فهم هيكل النموذج وعدد الباراميترات التي يحتاج النموذج لتعلمها خلال عملية التدريب.

# النتائج والتقييم لكال الشبكتين:

الشبكة الأولى:

* بالنسبة ل backpropagation :
  + loss: تتراوح بين 1.7 و 1.6.
  + accuracy : تتراوح بين 15.61% و 17.50.
* بالنسبة ل Rprop :
  + loss: تتراوح بين 1.8 و 1.5.
  + accuracy : تتراوح بين% 17.61و% 20.50.
* بالنسبة ل Bayesian :
  + loss: تتراوح بين 1.8 و 1.5.
  + accuracy : تتراوح بين% 17.61و% 22.50.

الشبكة الثانية:

قمنا بتدريب الشبكة على 10 epochs و batch\_size 256 وكانت النتائج على عينات التدريب:

* loss: 1.40
* accuracy : 0.448

وعلى عينات الاختبار :

* loss: 1.46
* accuracy : 0.426

ثم دربناها على 20 epochs و batch\_size 256 وكانت النتائج على عينات التدريب:

* loss: 0.79
* accuracy : 0.693

وعلى عينات الاختبار :

* loss: 1.66
* accuracy : 0.476

ثم دربناها على 10 epochs و batch\_size 128 وكانت النتائج على عينات التدريب:

* loss: 1.05
* accuracy : 0.593

وعلى عينات الاختبار :

* loss: 1.39
* accuracy : 0.476

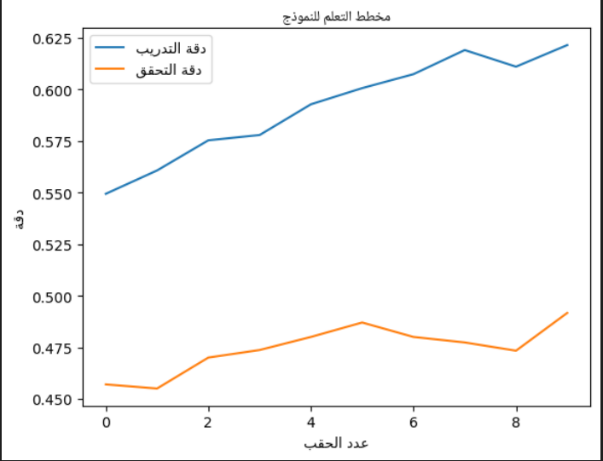
وبقينا ندرب دون تحسن كبير بالنتائج حيث أن أفضل نتيجة حصلنا عليها هي accuracy : 0.50

التقييم :

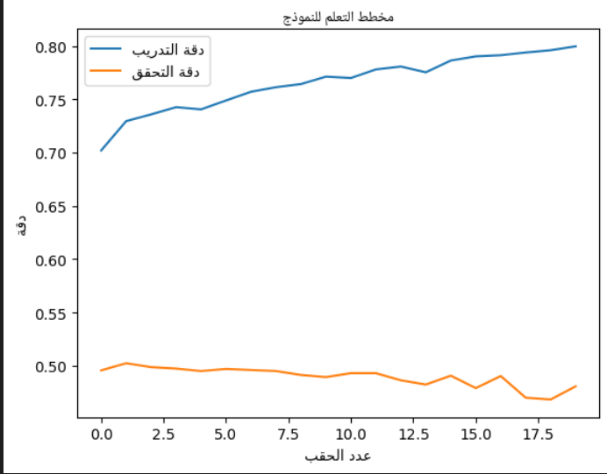
الشبكة الثانية أفضل لكن فعلياً الشبكتان تعطي نتائج سيئه في عملية تصنيف الصور .

# مقارنة نتائج التجارب

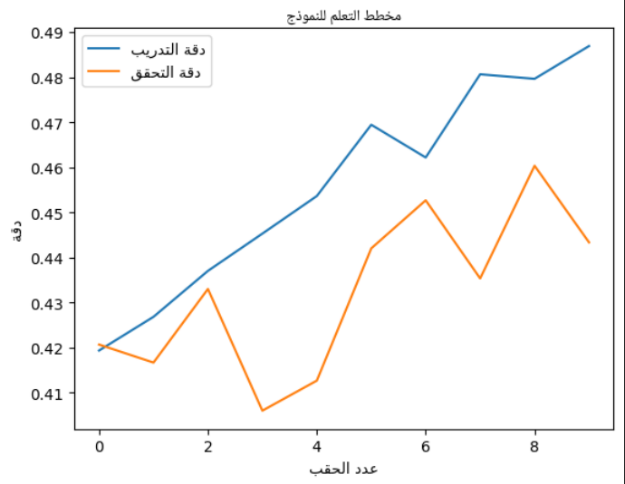
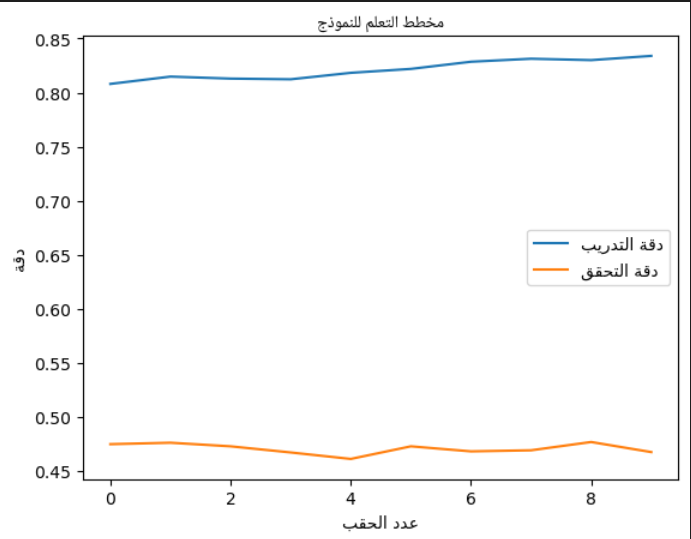
* التعديل على batch\_size : حيث انها تحدد حجم المعلومات الداخلة إلى الشبكة التي على أثرها نقوم بتعديل الأوزان .



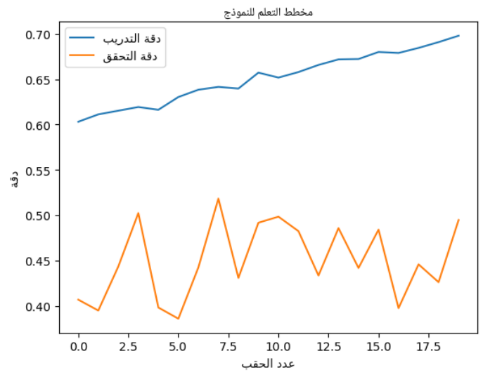
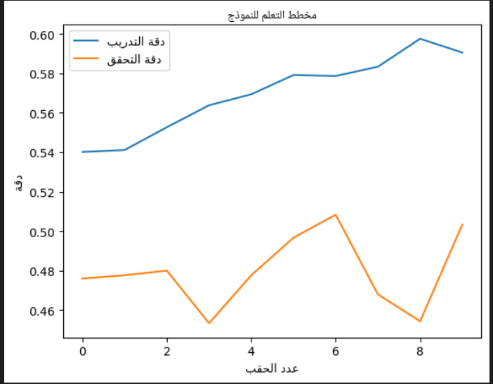
* التعديل على epochs : حيث قمنا بزيادة عددها من 10 إلى 20 ولو زدن أكثر لن يكون هناك تحسن بنتائج بالنسبة للمجموعة الاختبارو لكن سنجد فرق بالنسبة لمجموعة التدريب لان النوذج يكون قد قام بحفظ البيانات وهذا يأدي إلى مشكلة overfit وقد قمنا بمعالجة هذه المشكلة وسوف نقوم بشرح الطريقة لاحقاً.

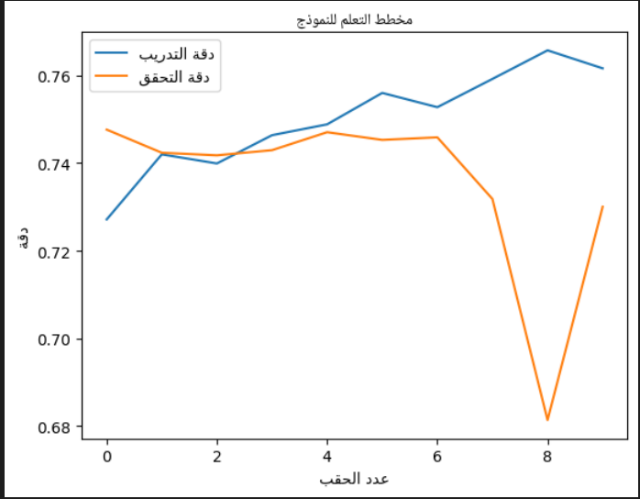


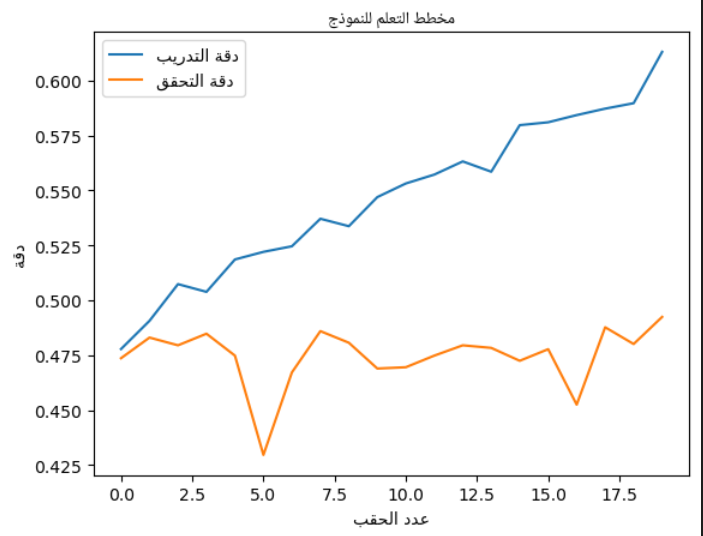
* التعديل على optimizer : حيث قمنا في البداية بالتعديل على البارامترات الأساسية في adam حيث قمنا بالتعديل على المعدل التعلم ووجدنا تحسن على بيانات التدريب ودون اثر يذكر على بيانات الاختبار فقمنا بتعديل أخرايضا علا معدل التعلم وايضأ لم نجد أي أختلاف يلحظ .



* تغيير ال optimizer : قمنا بتغييره من adam إلى sgd ولكن لم يقدم لنا نتائج أفضل من السابق .



* وقمنا بعدت عمليات تغيير على حجم الصور من زيادة في أبعاد الصور ولكن كانت زيادة أبعاد الصورة ترتد علينا بسوء النتائج وزيادة وقت التدريب وفي النهاية قمنا بالتثبيت على 128\*128 ك حجم للصور بحيث لا تكون قد سائت جداً دقت الصور رغم انه كان تتحسن قليلاً النتائج لتصبح ما بين 52 و 54 عندما نقوم بتخفيض أبعاد الصورة لكن لاحظنا أن الصور تصبح من دون ملامح .
* و في النهاية قمنا بتغيير توزع البيانات بحيث أصبحت 15330 صورة للتدريب و 1704 صورة للاختبار وقد لاحظنا تحسن ملحوظ بالنتائج حيث ابقينا optimizer sgd وقد كان accuracy 0.73 على بيانات الاختبار. 
* و بعد تغير حجم الداتا أعدنا المحاولة مع optimizer adam



Underfitting : الشبكة لا تتعلم الأنماط الكافية من البيانات، مما يؤدي إلى أداء ضعيف على بيانات التدريب والاختبارمما يؤدي إلى دقة متدنية على كل من بيانات التدريب والتحقق.

ولتجنبها نقوم بزيادة عدد العقد والطبقات وزيادة عدد الحقب..

Overfitting : الشبكة تتعلم الكثير من التفاصيل والضوضاء في بيانات التدريب، مما يؤدي إلى أداء ضعيف على بيانات جديدة مما يؤدي إلى دقة عالية على بيانات التدريب لكن دقة متدنية على بيانات التحقق أو الاختبار.

ولتجنبها نقوم بإزالة بعض الطبقات أو العقد واستخدام طبقة Dropout لتقليل الاعتماد على عقد محددة وتعزيز البيانات أو جمع المزيد من البيانات و التوقف عن التدريب عندما لا تتحسن دقة التحقق.

# المصادر